NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THE FORMAL SPECIFICATION OF
COMPUTER SYSTEMS

by

Klaus Karrasch

December 1987

Thesis Advisor: Daniel Davis

REPORT DOCUMENTATION PAGE

PORT SECURITY CLASSIFICATION :lassified	1b RESTRICTIVE MARKINGS
CURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; Distribution is unlimited.
ECLASSIFICATION / DOWNGRADING SCHEDULE	
FORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
AME OF PERFORMING ORGANIZATION .1 Postgraduate School	6b OFFICE SYMBOL (If applicable) Code 52
	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
DRESS (City, State, and ZIP Code) .erey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
ME OF FUNDING / SPONSORING GANIZATION	8b OFFICE SYMBOL (If applicable)
	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
DRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

E (Include Security Classification)
FORMAL SPECIFICATION OF COMPUTER SYSTEMS USING PETRI NETS (u)

SONAL AUTHOR(S)
asch, Klaus

PE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
er's Thesis	FROM _____ TO _____	1987 December	129

PLEMENTARY NOTATION

COSATI CODES		
LD	GROUP	SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Formal Specification, Petri net, Abstraction of Computer Resources, Timing of Computer Systems

TRACT (Continue on reverse if necessary and identify by block number)

With the introduction of formal specification of abstracted computer resources, both physical and logical, there is the possibility that a major step forward can be made toward developing a methodology for reducing the portability and reusability costs of computing system components. Still, the current methodology is only concerned

RIBUTION / AVAILABILITY OF ABSTRACT CLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
AE OF RESPONSIBLE INDIVIDUAL E. Daniel Davis	22b TELEPHONE (Include Area Code) (408) 646-3091 / 22c OFFICE SYMBOL Code 52Dv

M 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19. ABSTRACT (continued)

with the static functional properties of resources and not their timing properties. This places limitations on the generality of the method. This study describes a way to formally specify the timing of computer systems by combining ideas of both semantic algebras and Petri Nets.

Approved for public release; distribution is unlimited.

The Formal Specification of Computer Systems using Petri Nets

by

Klaus Karrasch
Lieutenant, Federal German Navy

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE
from the

NAVAL POSTGRADUATE SCHOOL

December 1987

ABSTRACT

With the introduction of formal specification of abstracted computer resources, both physical and logical, there is the possibility that a major step forward can be made toward developing a methodology for reducing the portability and reusability costs of computing system components. Still, the current methodology is only concerned with the static functional properties of resources and not their timing properties. This places limitations on the generality of the method. This study describes a way to formally specify the timing of computer systems by combining ideas of both semantic algebras and Petri Nets.

TABLE OF CONTENTS

I. INTRODUCTION

Timing in computer systems has been a critical issue throughout the evolution of computers. The most obvious areas where timing is of great concern are operating systems, distributed systems and real-time systems. In the most other areas, timing of a computer program is taken for granted since we assume a simple sequential execution of this program. But we have to realize that, when we consider the program and the machine on which it has to run as a whole, i.e. a computer system, we have to deal also with the internal timing of the hardware. Even though high level programming languages have provided us with the means to software from one system to another, we still find these programs may not work properly because of timing problems. These timing problems are normally caused by different implementations of computer resources. So It would be very helpful to have a way of comparing computer systems and predicting problems in timing when programs are transferred. It would even be much better to have specifications of computer systems that could be used to design transferrable programs in the first place.

There is ongoing research at the Naval Postgraduate School on the formal specifications of computer systems that is mainly intended to overcome the increasing costs of

computer software resulting from problems with portability and reusability of programs. The first result of this research project has been the development of a formal specification methodology by Davis (1984). This methodology was successfully used to write a formal specification of an Abstract Processor by Yurchak (1984).[1] This work was followed by several extensions of the Abstract Processor and related work by Hunter (1985) and Zang (1985). The research performed at the Naval Postgraduate School is part of a relatively new branch of computer science: the **Science of Computing System Design** which is concerned with a formal approach to the specification and description of computer systems. This thesis follows the direction of previous work done in this field. Its objective is to formally specify timing in computer systems. Even though there has been considerable interest in timing, our approach will emphasize two aspects of the problem:

- We want to develop a formal way to specify timing to achieve the benefits of the rigorous foundation of a formal description.
- We want to specify abstracted resources which include both hardware and software with a unified approach.

Throughout this thesis the term "computer resources" is used in a sense that combines all hardware and software building blocks of computer systems: memory, registers, data types, instructions, etc.. Also the special aspect we are

---

[1] An edited version of the specification of the Abstract Processor is included in Appendix A

concerned with is time as a computer resource. Computer resources can be either pure physical or they can be an abstract type. A memory cell belongs to the physical category while a specific data type belongs to the abstract category. The means to deal with these differences is abstraction. Specifying computer resources as abstractions in a mathematical way also allows us to compare different specifications and implementations.

Within this work we will emphasize a practical viewpoint. Even though in applying pure mathematical concepts to computer systems we realize that computers are in no way ideal: as every piece of hardware is finite (especially the memory) and an event in the computer is never instantaneous but will take a certain amount of time. In this context, for example, the term "digital computer" is misleading since there are many undefined states between those defined "0" and "1".

Therefore the goal of this thesis is to provide a methodology for the rigorous specification of timing:

- to evaluate the time behavior of systems which leads to the exhibition of possible places in system where parallelism could be used,
- to give a better understanding of the dynamic aspect of computer resources in time, and
- to find time critical situations in a system.

II. BACKGROUND

A. FORMAL SPECIFICATION

The idea of specifying a system formally is to deal with physical and abstract computer resources as abstractions to support our major concerns with **portability** and **reusability**. In this sense we deal with computer resources as abstractions which we want to describe such that the functions and the properties of a resource are stated in a mathematical way to support precision and provability. Studies in this direction have been conducted by many researchers (Goguen (1978), Guttag (1978), Bergstra (1983), and Davis (1984)). As a short introduction to this work we would like to present some key issues here as they were used in the first formal specification of the Abstract Processor.

The formal specification of the Abstract Processor is based on the method of algebraic specification which consists of two parts: the interface specification and the constraints specification. The interface specification declares operands and the operators that can be applied to them, so information for syntactic constructions and type checking are provided. The constraints specification is a set of properties that define constraints on the operations. These properties are stated by equations that associate the same meaning to pairs of expressions of the specification. As an example the

specification of the computer resource "boolean" is shown in Figure 2.1.

```
Resource boolean is

    Operands
        bool

    Operators
        not: bool -> bool;
        and: bool,bool -> bool;
        true: -> bool;
        false: -> bool;

    Properties
        not(true()) = false();
        not(not(x)) = x;
        and(true(),x) = x;
        and(false(),x) = false();

end boolean;
```

Figure 2.1: Specification for Resource "boolean"

Figure 2.1 illustrates the definition of one operand type (bool) and the operations (not, and, true, false) that are allowed with this operand. The operations are stated as functions with their input and output. Note here that the constants resembling "true" and "false" are obtained from the nullary operator functions with no input ("true()" and "false()"). Up to this point only the interface part is considered. The meaning of the specification is indirectly in the form of equations that state that certain expressions must be treated identically to other expressions. The above equations use "x" as a free variable, i.e. "x" stands for any

expression that can represent an operand of type "bool". So far this computer resource "boolean" is an abstract data type in the traditional meaning. But computer resources also consists of physical resources which are very similar to abstract data types in their specification. Figure 2.2 provides an example of specifying a physical computer resource to indicate the memory state of the Abstract Processor. For simplicity only the operands for initialization, fetching and storing are presented. The first interesting fact to note in this specification is that it states that the resource "amstate" is an extension of the previously defined specifications of the resources "boolean", "memaddress", and "regaddress", i.e. all operands, operators, and properties defined in these specifications can be used to specify "amstate" without further explanations. The operand "state" has in this example four operators to initialize the processor, to fetch from memory and registers, and to store to memory and registers. The properties for these operations are shown by equations that indicate their relations among them. Note that this example uses the term "undefined" to indicate an error or don't care condition (the attempt to fetch the contents of an register or memory address of a new initialized processor is illegal).

The basic step in becoming familiar with formal specifications is to consider the well-known constructs of abstract data types: a class of objects together with a set

of operations that may be applied to these objects. This approach can also be applied to physical computer resources.

```
Resource amstate is

    Extension of
        boolean,
        memaddress,
        regaddress,

    Operands
        state;

    Operators
        fetchm: memaddr,state -> val;
        fetchr: regaddr,state -> val;
        storem: val,memaddr,state -> state;
        storer: val,regaddr,state -> state;
        initam: -> state;

    Properties

        fetchm(a,initam()) is undefined;
        storem(fetchm(a,q),a,q) = q;
        implies(eqmemaddr(a1,a2),
            fetchm(a1,storem(v,a2,q)) = v)
            = true();
        implies(not(eqmemaddr(a1,a2),
            fetchm(a1,storem(v,a2,q)) = fetchm(a1,q))
            = true();

        fetchr(r,initam()) is undefined;
        storer(fetchr(r,q),r,q) = q;
        implies(eqregaddr(r1,r2),
            fetchr(r1,storer(v,r2,q)) = v)
            = true();
        implies(not(eqregaddr(r1,r2),
            fetchr(r1,storer(v,r2,q)) = fetchr(r1,q))
            = true();

end amstate;
```

Figure 2.2: Specification of "amstate"

Then we have concrete algebras which describe an aggregate of operations and sets of values where the sets are the source for arguments and result types of each operation. This is a system in which there are sets and operations that are applied to elements of the sets such that the results of the operators stay in the system. When we construct a specification of such a system, we attempt to create a specification that serves as templates for the sets and operators in a concrete algebra and axioms which state provable equations about the values and operations. With such a specification we have something which describes the resource abstractly and precisely without restricting it to a specific concrete algebra, i.e. there can be many algebras that are implementations of a single specification. They are considered as the class of algebras uniquely associated to that specification.

1. Syntax versus Semantics

We refer to the syntax of description as the "form" of the description and to semantics as the "meaning" of the description.

The meaning is always determined by associating form to real objects. Basically the syntactic part describes legal expressions that can be formed with the operators in the specification. These expressions are called **formal terms**. The constraint part specifies that certain formal terms are to be considered equivalent. The meaning of specifications is

established by associating certain concrete algebras to the specification. There algebras represent the "real object". Operational expressions in the concrete algebras are called **actual terms**. Semantics are defined by a correspondence between properties of formal terms and actual terms. A real object is a realization of the abstract object defined by a specification under three conditions as they are stated in Davis and Yurchak (1984):

- Condition 1:
  For each operand type of the specification there is a corresponding set of values in the real object and to each operator in the specification there is a corresponding operation in the real object that is defined on values that correspond to the operand types of the operator.

- Condition 2:
  In the correspondence between formal terms and actual terms, two formal terms are provably equal if and only if their corresponding actual terms have the same value.

- Condition 3:
  To every value of the real object there must correspond some formal term whose corresponding actual term has that value.

These conditions provide us with a powerful insight: given a formal specification of some resources there can be different implementation in the real world (as they probably are on different machines), but as long as the implementations satisfy the above conditions they are equivalent. This is a very important property when the issue of portability is concerned.

Still an formal specification despite its abstract view of resources has to deal with the real world: for

example there is nothing like true infinite memory so that a defined operator like "nextmemaddr" (to obtain the memory address of the next instruction to be executed) will eventually exceed the physical implemented memory of a system. The "undefined" has been introduced in the formal specifications to act like a safeguard. It indicates that there is no interpretation in the realization for this term.

B. PETRI NETS

Petri Nets are tools for the study of systems through modeling. The Petri Net theory has been originally introduced by Carl Adam Petri in his doctoral dissertation (1962). Further studies of A. W. Hold and Jack B. Dennis helped to develop this theory.

1. Terminology of Petri Nets

From Peterson (1981) a basic Petri Net is defined as a five-tuple M = (P,T,I,O,m) which is composed out of the following parts:

- a set of places P
- a set of transitions T
- an input function I
- an output function O
- a marking vector m

The function I is a mapping from a transition t_j to a collection of input places $O(t_j)$ and the function O is a mapping of a transition t_j to a collection of output places $I(t_j)$, the marking vector m indicates the number of tokens

preset in each place. In this general form, developed by C.A. Petri, a place can hold more than one token at a time.[2]

As an example, the following net structure is presented here and Figure 2.3 shows its corresponding graph in the common symbology of Petri Net graphs where circles indicate places, bars indicate transitions, and arrows show the connections between places and transitions:

$M = (P, T, I, O, m)$
$P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$
$T = \{t_1, t_2, t_3, t_4, t_5\}$
$m = (1,0,0,0,0,0,0,0)$

$I(t_1) = \{p_1\}$	$O(t_1) = \{p_2, p_4\}$
$I(t_2) = \{p_1\}$	$O(t_2) = \{p_3\}$
$I(t_3) = \{p_2, p_5\}$	$O(t_3) = \{p_5, p_6\}$
$I(t_4) = \{p_4, p_5\}$	$O(t_4) = \{p_7\}$
$I(t_5) = \{p_6, p_7\}$	$O(t_5) = \{p_8\}$

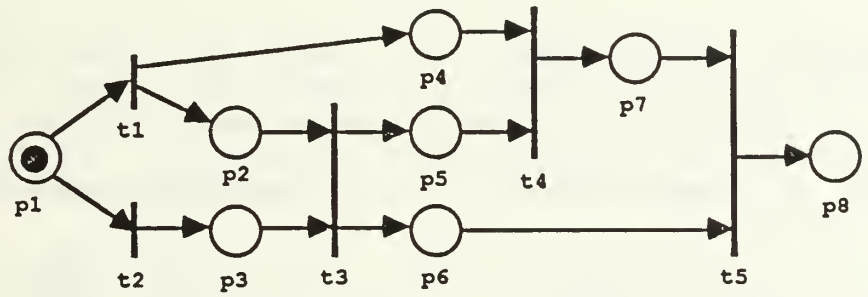

Figure 2.3: Graph of Petri Net

The following Petri Net terminology is used in this thesis:

- place = a construct for modeling conditions of the system
- event = actions that take place in the system
- token = a construct used to indicate that a condition holds (a true condition)

[2] The reader is referred here to the BAG theory

- **transition** = the process of recognizing true preconditions, the occurrence of an event, and making the postconditions hold
- **concurrency** = two or more events depending on different preconditions can occur in any order
- **conflict** = only one of two or more events depending on at least one common precondition can occur

To give a simple description of Petri Nets we can say the following: An event occurs (a transition is initiated or enabled) when all of its preconditions hold. The effect of the occurrence is that the tokens of the preconditions are "used" for the event and then distributed to the postconditions.

The following constructors can be recognized in Petri Nets:

- **simple transitions** = there is one precondition and whenever this condition holds the event occurs so that the token from the precondition is removed and after the occurrence of the event the token is moved to the postcondition so that this condition now holds (Figure 2.4).

- **conjunctive transitions** = there are two or more preconditions that all have to hold in order for the event to occur. All tokens of the preconditions are used and after the occurrence of the event only one token is moved to the postcondition to indicate that it now holds (Figure 2.5).

- **disjunctive transitions** = one precondition is connected to two or more events and when this precondition holds one of the events will occur and will move the token from the precondition to the postcondition of that event that occurred. The selection of the event to occur is non=deterministic (Figure 2.6).

- **distributive transitions** = there is one precondition and when this holds the connected event will occur. It will remove the token from the precondition and it to all postconditions so that every postcondition of this event will have a token (Figure 2.7).

- **complex transitions** = combinations of the above simple constructs of Petri Nets.

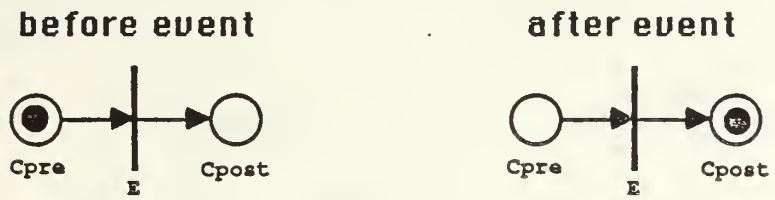

Figure 2.4: Simple Transition

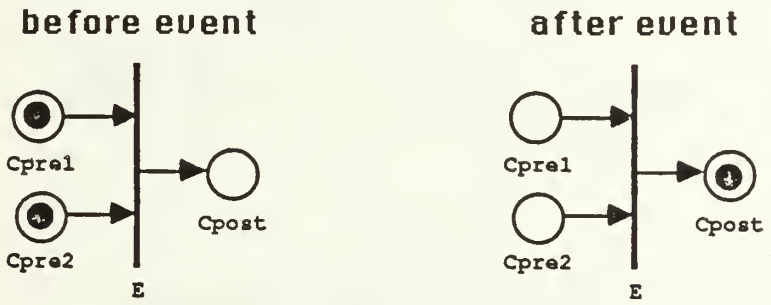

Figure 2.5: Conjunctive Transition

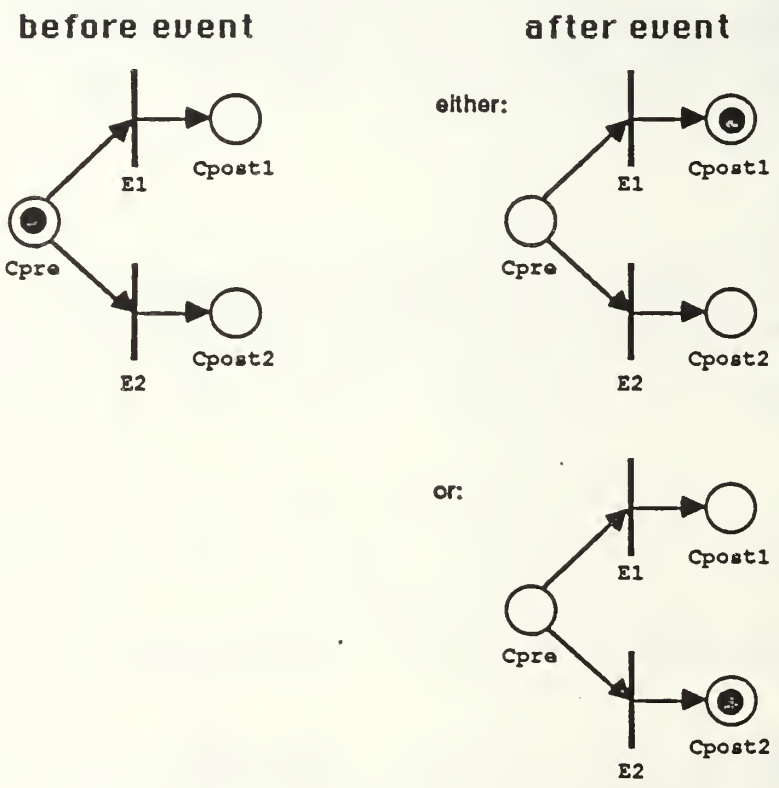

Figure 2.6: Disjunctive Transition

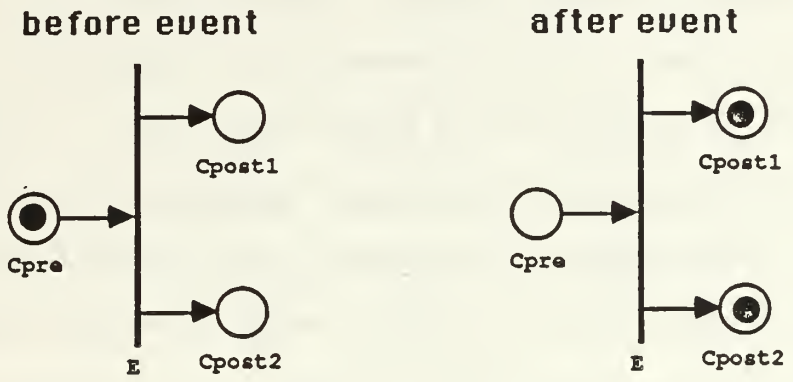

Figure 2.7: Distributive Transition

2. Properties of Petri Nets

Petri Nets by their nature are well suited to model asynchronuous processes, i.e. where the progress of a process is controlled by conditions and events and not by some kind of fixed clock. This means that in some part of the net there can be waiting for a condition to start an event, even the case that a process cannot continue because of a missing condition. Suppose an event is modeled as a conjunctive transition as indicated in Figure 2.5. If the precondition C_{pre} is never true the process will stop at this point. The "flow" of the progress can always be observed by the state of the condition places in the system. The occurrence of events is recognizable by the changing of the preconditions and postconditions.

The discussion of disjunctive events has shown an important property of Petri Nets: their non-deterministic nature, i.e. we have under normal circumstances to force a distributive construct in one or the other direction. This is a major obstacle when we have to model some kind of decision making in a Petri Net. One way to get around this problem is the construct (introduced by Peterson (1981)) of an "external agent" which provides appropriate TRUE or FALSE places at decision points in the net. Here by the intervention of the "external agent" the process proceeds in the direction of the TRUE or FALSE place. In general, one can think of "external agents" as CASE-constructs in high-level programming languages such that, dependent on the value of an argument, one and only one action is taken by setting the according place in the net. We will discuss this topic further in Chapter IV.

When we look at the conjunctive and distributive constructs we observe that by combining them we can build a distributive construct which "fans" out into several holding places and then combines again with a conjunctive construct. In this way, we have able to introduce **parallelism** into Petri Nets. Figure 2.8 shows the graph of a process that fans out into five processes which then merge again into one process. This capability of Petri Nets is very powerful and convenient in specifying computer resources.

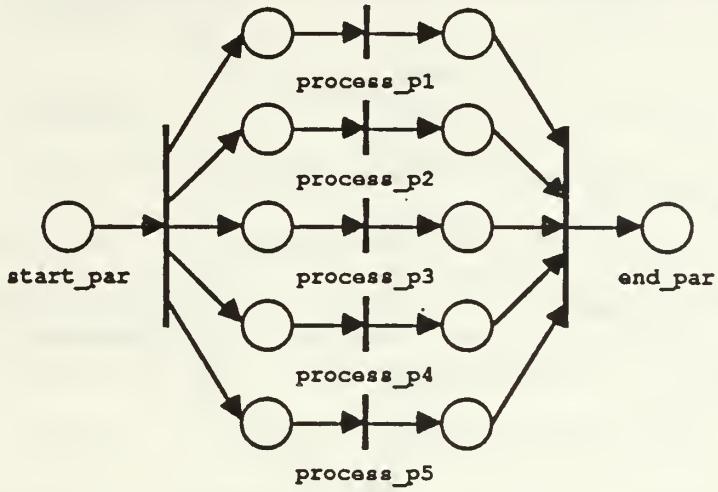

Figure 2.8: Parallelism with Petri Nets

3. Modeling with Petri Nets

Modeling with Petri Nets has been widely used in very different areas: computer software, computer hardware, chemical reactions, queuing theory, political systems, etc.. Two examples as they are presented by Peterson (1981) are given to illustrate this modeling work: a portion of a Petri Net showing a control unit of a computer with multiple registers and functional units as an example for modeling computer hardware (Figure 2.9) and a Petri Net dealing with the mutual exclusion problem as example for modeling computer software (Figure 2.10).

In our approach we want to exploit the ease and the properties of Petri Nets for modeling the combination of computer hardware and software as they operate in time.

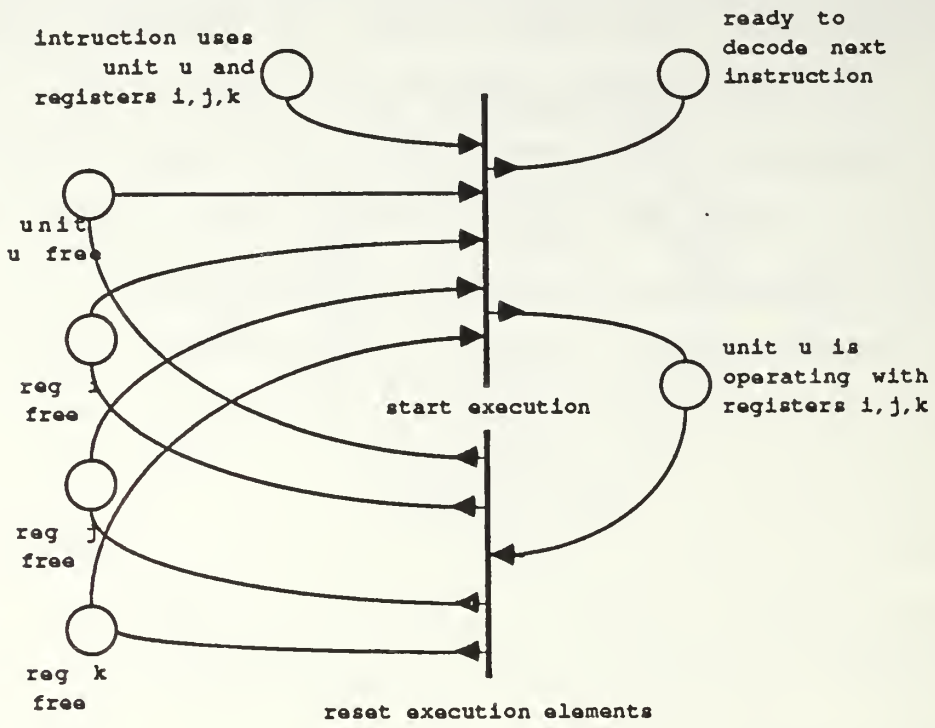

Figure 2.9: Computer Control Unit

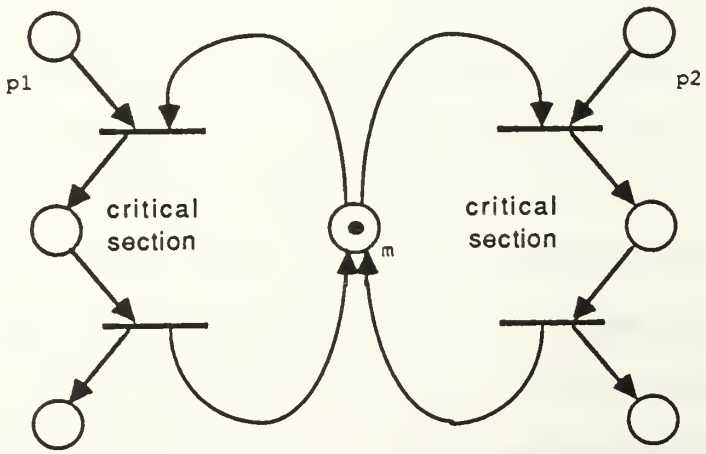

Figure 2.10: Mutual Exclusion

III. THE PROBLEM OF TIMING SPECIFICATION

A. GENERAL PROBLEMS

Why are we so concerned about the timing of a system? Time is an important resource in computer systems that must be managed carefully. There is almost always the possibility that if we invest more of other resources we are able to reduce the amount of time a piece of work will need. Everybody remembers a mathematical problem like this: if it takes one unit to accomplish a task in x hours how many hours will it take y units to do the same task? In this very simple problem the increase of other resources (e.g. units) will reduce the needed time for the task linearly. In computer systems we could save time by implementing more CPUs, disk drives, arithmetic units, etc.. But since more hardware costs more money and the control of additional resources uses time by itself we have to be very careful in determining the resources we need and how we use them. The following example shows the danger of mismanaging computer resources: a computer task that requires some resources (e.g. disks) would waste them if it holds more than it needed at times when it actually does not need them and so prohibits other tasks from using them.

The formal specification as described in the specification of the Abstract Processor is only concerned

with static computer resources, i.e the timing properties are implied by the functional relations between components of the system. The static specification is purely functional. For example, operands must be evaluated before a function is applied but there is no way of indicating the order of evaluation. The dynamic computer resources are those that express an ordering of resources in time, mutual exclusion and concurrency. Instead of assuming some ordering in the use of computer resources we want to be able to explicitly state and define the timing of a system.

The goal is to specify the required timing properties precisely to a desired degree which for example is sufficient to evaluate the system for time and cost efficiency. The relation between time and cost depends on the nature of the system: there is much more emphasis on time in system that are very time critical (e.g. real-time systems) and not so much on systems that are purely problem solvers.

The basis of this work is to show whether such a methodology for specifying timing properties can be based on the theory of Petri Nets and how well the special cases of timing in computer system resources can be expressed in terms of Petri Nets.

B. SPECIFIC PROBLEMS OF INTEREST

1. Order of Evaluations of Functions

Given a function $f(x_1, x_2, \ldots, x_n)$ we only require that x_1 to x_n are evaluated before f can be applied.[3] There is no statement that the evaluation of x_i has to be started first or what evaluation has to be completed first.

2. Parallel Processing of Parts of Functions

Considering again our function $f(x_1, x_2, \ldots, x_n)$ we want to state explicitly which evaluations have to performed in parallel and which in sequence. Why do we want to do so? Following our purpose, in the specification we want to describe timing in a way which is as exact and detailed as a timing diagram used to construct hardware.

3. Mutual Exclusion

A major problem that arises with parallelism is mutual exclusion, i.e. a computer resource can only be used by one process at a time and the use of the resource has to have a certain entry and exit point to preserve the integrity of the resource.

Consider a simple computer with a memory unit which retrieves and stores data on request via a specified interface (Memory Buffer Register and Memory Address Register). This is parallelism even in simple computers since the memory is independent from the CPU. So we have to make

---

[3] Even if x is a constant it has still to be evaluated, i.e. its value has to be retrieved

sure that only one request is handled by the memory unit at a time and that the next one is handled when the first one is finished. We want to be able to explicitly state those properties in the specification of timing systems.

4. Data Flow

During the course of a process in time certain data has to be available in order for the process to proceed. Some data is changed, other data is not needed anymore. Here we have the problem of how to model data flow by means of Petri Nets.

To make this point more clear let us consider the execution of following instruction: SUB R1,R2 (subtract the contents of register 1 from the contents of register 2 and store the result in register 2). During the execution we have to retrieve the identities of the registers from the instruction (i.e. the instruction has to be decoded) then their contents both have to be available before we can perform the subtraction. At this point of the execution we do not need the identity of the first register anymore, but the one for the second since it is not only a source for the operation but also the destination. Thus, we have to have some mechanism in our methodology to state data which is available during the course of the execution.

IV. APPLICATIONS OF PETRI NETS TO THE TIMING IN FORMAL SPECIFICATIONS

In previous work computer resources have been formally specified using basically the algebraic specification approach. In essence, this is a type of functional specification. The question we address here is can functional specification be extended, using Petri Net theory, to provide for the specification of timing properties.

A. GENERAL APPROACH

Given a general function $f(x_1, x_2, \ldots, x_n)$ what are the stages of evaluation for this function?

- the evaluation of $f(x_1, x_2, \ldots, x_n)$ must have been requested from somewhere and by this the evaluation gets into a **requested** stage
- for all x_i, $1 <= i <= n$, the evaluation is requested which starts for all x_i a new process with the same stages as described here
- once all x_i are evaluated and their results are available to our function f it can be evaluated in a **processing** stage
- when the evaluation is completed and the result is available the process is **completed**

Note the similarity to the "natural" way a human being would calculate this function: if we were to calculate *sum(sin(x²),sqrt(y))* we had to calculate the square root of y and we had to square x and take the sine of it and then we would apply the sum-function to the intermediate results.

However, this example exhibits a problem for our very general approach: how do we know what the values of x and y are and how do we know that e.g. "sqrt" means "take the square root". Therefore there must be some decoding and retrieval steps in between which determine what the parts of the function expression mean. This is exactly the case when we consider computer instructions: e.g. given an instruction like "ADD R1 M2 R3" which means "add the contents of register 1 to the contents of memory address 2 and store the result in register 3". Here the following steps have to be performed:

- The instruction has to be decoded (we assume that at this stage the instruction is already retrieved): i.e. the components of the instruction (operator and operands) must be made available to the further evaluation.
- Up to this stage only the names (i.e. the symbolic addresses) of the operands are available and so the next step is to retrieve the values of the operands.
- Now that the operator and the values of the operands are available the operation designated by the operator can be performed.
- When the result is available it can be stored into the location which expressed by the third operand.

Figure 4.1 shows the corresponding graph of a Petri Net describing the above steps. In this example we see an approach to describe the execution of an instruction in a sequential manner. Suppose we had a machine that could perform retrieval of values and operation in parallels, how could we describe that certain steps could be performed in parallel and how could we mark points in time where the process can only proceed if some results are available?

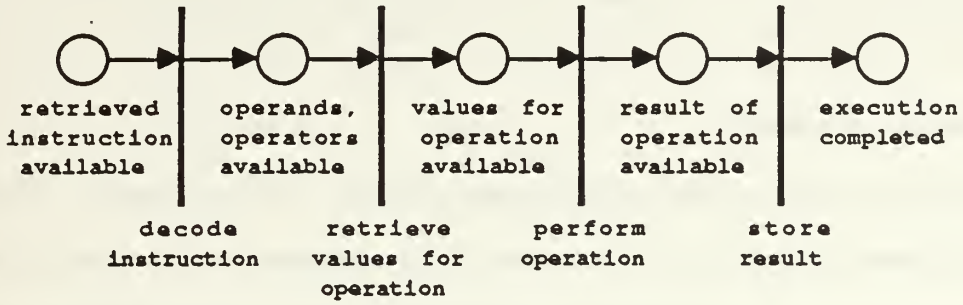

Figure 4.1: Simple Instruction Execution Net

This is the point where we can use the properties of Petri Nets: if we model requests and availabilities as places of Petri Nets and the actions on requests and availabilities as transitions and connect them accordingly we are in a position to model the evaluation of a function or the execution of an instruction.

Up to this point there is nothing new in our methodology since modeling with Petri Nets is common practice and has been done for a long time. The question to be ask now is does a methodology based on Petri Nets provide the means to specify the specific problems of computer systems and their components in a way that is consistent with the formal specification of the static properties we have seen in the specification of the Abstract Processor.

In addition we not only want to look at the timing of systems in an isolated fashion, but also we want to combine the specification of static properties, as introduced in the

specification of the Abstract Processor, with the specification of dynamic properties of such a system. With this combination we can specify systems completely.

B. NOTION OF SUBNETS

Now that we have Petri Nets as a tool we can model simple timing systems using our methodology. However, we would like to simplify and eliminate redundancy: if we use the same structure in a net several times, it would be better to have this structure defined once and reference this definition wherever we need it in our system (Figure 4.2). As an example, suppose we realize that a structure to make the contents of a certain memory address available to the process appears several times in our system. By defining this retrieval-function as a subnet we are in a position to use it everywhere in the system simply by setting its ENTRY-place (in our example with a request for value of a specified register) and we obtain the result (the value of the specified register) at its EXIT-place. This works even for the case that the subnet specifies a computer resource that has to be accessed observing mutual exclusion.

The major question that has to be asked here is how can we model such a subnet that has the ability to "sense" where it has been invoked and so can return its results to that location in the system. Computer language constructs like procedures or functions use a return address which is saved with the call of the procedure/function to determine the

location in the program which has to be executed after the procedure/function is finished. Our model of using Petri Nets is different in this aspect: despite the fact that it shows the dynamic behavior of a system, its structure is static and does not change in time and so all connections between nets and subnets are established. Since we have to know all these

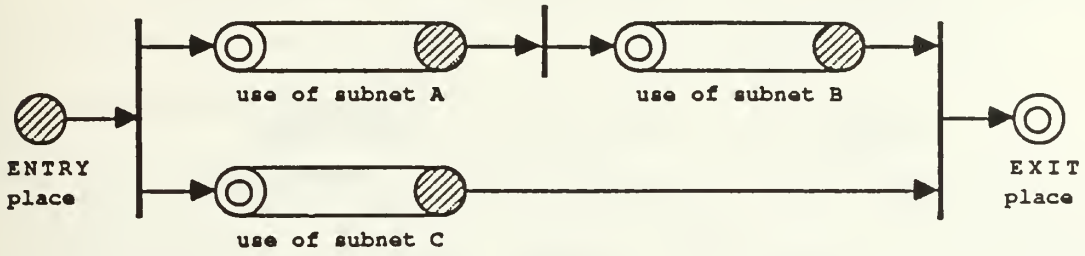

Figure 4.2: Symbology of Subnets

connections we are able to provide for each connection an entry-place which is connected to the net internally by disjunctive events such that only one can be "fired" at a time. The "firing" of those events lets a path-place hold that indicates the entry-place which triggered the event. This path-place decides what exit-place is set when the internal net provides the result.

This definition of a subnet is a very powerful shortcut for keeping descriptions of systems limited. It resembles a function construct in a high-level programming language: it is been "called" by setting its request-place, does its

supposed work and "returns" the result as the available-place.

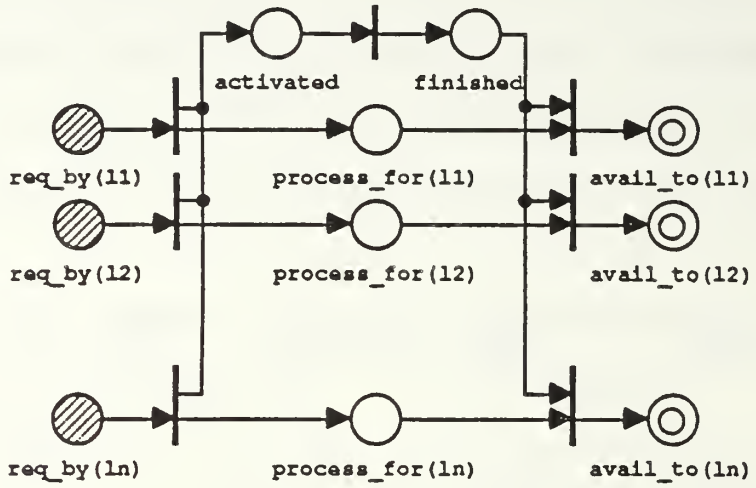

Figure 4.3: Generalized Subnet without Mutual Exclusion

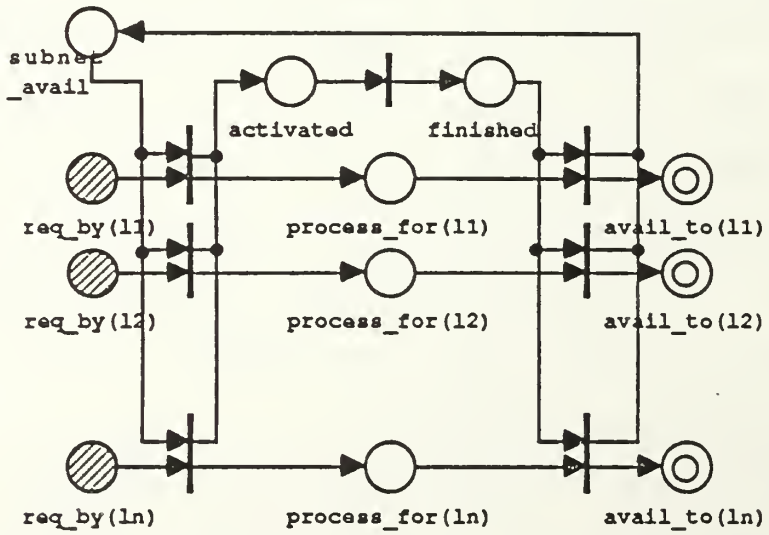

Figure 4.4: Generalized Subnet with Mutual Exclusion

We have to show that our methodology is capable of defining subnets and introducing them into other nets.

C. COMBINING NETS

With the introduction of subnets we are in need of rules and guidelines on how we can combine a collection of small nets and subnets into a timing system.

1. Coupling of Nets

In the preceding paragraph we have assumed that nets are connected by entry and exit places, but generally, we have two possibilities to connect nets:

- Coupling by places: to connect two nets the first net outputs to a EXIT-place that is read by the second net as an ENTRY-place. In the special case that we use subnets in our specification we request the subnet by place (the ENTRY-place of the subnet) and obtain the result by a place (the EXIT-place of the subnet).

- Coupling by events: events are shared between nets and when the ENTRY-event "fires" the requested net is invoked and signals the availability of the result by "firing" its EXIT-event.

We have chosen the coupling of nets by places because of the following reasons:

- The request for a subnet submitted by a place allows the requested subnet more "liberty" to react on the request only when it is ready to do so since the pending request as a "loaded" place remains until it is used by the net, where as in event-coupling the saving requests had to be done in a more complicated way.

- The coupling of nets by places resembles the way events like interrupts are processed in real machines, here the interrupt does not interrupt the execution of instructions at any time, but a status "interrupt" is set and this status is checked between the execution of instructions and acted upon.

2. Extensions of Nets

To make our methodology consistent we need to state what other nets we are going to use in this net. There is a close similarity to INCLUDE, USE or WITH constructs of high-level programming languages or the EXTEND construct of the formal specification. In the specification of timing the mentioning of a net to be the extension of another means that the parent net is going to use the extended net as a subnet in the specification.

3. Net Selection

Due to the non-deterministic nature of Petri Nets we do not have a traditional net construct which can make decisions on truth or falseness and directs the path in the net accordingly. A proposed solution for this problem by Peterson (1981) is to use "external agents" as they were presented in Chapter 2. This construct is able to examine a status or data on a given condition and make the decision whether the condition is fulfilled. With the outcome of the decision a path to a place representing true or the place representing false is set. By extending this idea we able to think of "external agents" as a CASE-statement where one and only way through the net is chosen according to a condition (see Figure 4.5).

D. TYPING OF NET ELEMENTS

In the traditional Petri Net theory we have tokens to indicate the flow through the net and to mark holding places.

So the "firing" of an event consists in collecting a token from each of the connected input places, performing the designated action and distributing a token to each connected output place. This is not enough when we want to describe the flow of data in time. Also we want to be able to state specifically what kinds of data, what types of data are being requested, available or transferred at a certain point in the timing of a system.

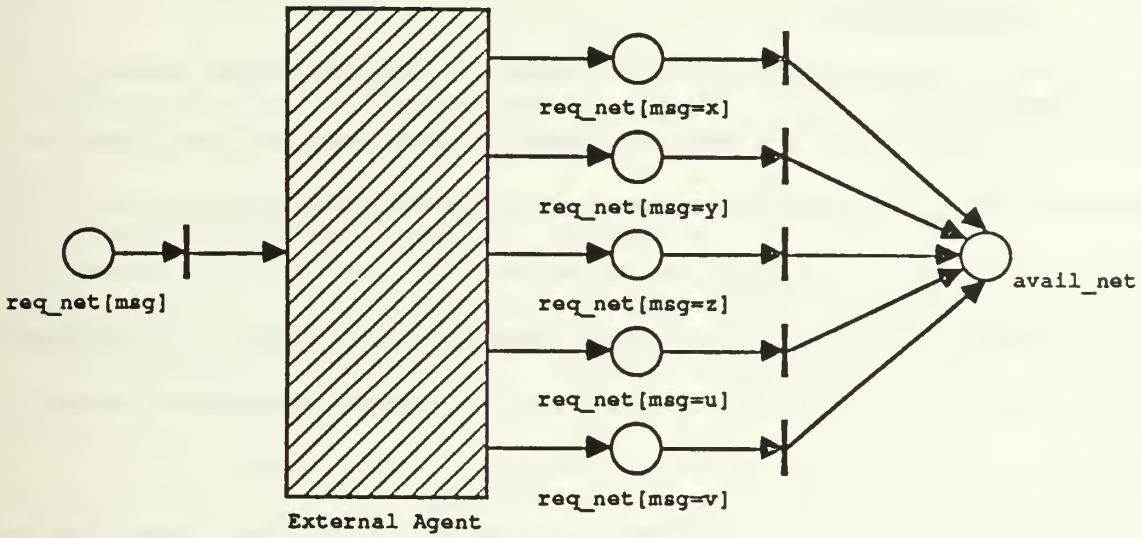

Figure 4.5: Net Selection by "external agents"

How can we show the presence of data in our methodology? On first sight we have two possibilities: either we introduce a typed place or a typed token. Both of these constructs could indicate the presence of certain data. So we to examine both methods under the consideration which of them suits our

goal better to develop a practical and understandable methodology.

1. Typed Places

When we introduce a concept of typed places we still have the original meaning of the tokens to indicate the holding of a place. The presence of data of a certain type must be accomplished by means that have to obey the type of the place. The events still react on the presence of tokens in the places.

2. Typed Tokens

This alternative considers a token as a construct that carries an actual piece of data according to the type of the token. We can imagine tokens as a message residing in the places. The events now can be modeled by picking up a message from each connected input placed, performing their designated actions, and distributing a message to every connected output place.

So now we can look at places in a net as constructs that can receive token-messages from events, keep them and send them to events. The actions performed by events consist of picking up message-tokens from places, changing and creating message-tokens and sending them to places.

3. Typed Tokens in Typed Places

Both concepts above exhibit some disadvantages:

- Pure place typing needs some external mechanism to establish the presence of data.

- Pure token typing allows a message-token to be sent to every place in the net since there no protection from receiving message-tokens of the wrong type.

This leads to the idea of combining both typing schemes. With this concept we restrict places in the net to accept only tokens of a certain type and the tokens are actually typed messages. One small problem does arise here: what if we want in some situations the token to have its original meaning only to indicate its presence without any data? This reminds of a message without contents. As a solution we introduce following typing convention:

- A place declared to be of a certain type or collection of types can only accommodate tokens that are messages of this type or collection of types.
- A place declared to be of no type can only accommodate tokens which are "empty" messages.
- An event will collect typed token-messages from the connected typed input-places, perform its designated action and output typed token-messages to the connected typed output-places.

With this typing scheme we are now in a situation to specify data flow in time by means of typed tokens and places. Although we have based our work on Petri Nets we see that our concept has become more general and now reminds of a general message passing system.

E. SYNTAX

Since our approach includes the existing specification methodology of the static computer resources, we have to carefully develop a new syntax which expresses both the static and dynamic properties of the systems to be specified.

The following syntax has been introduced with the specification of the Abstract Processor by Davis and Yurchak (1985):

```
Resource <name> is
    Operand Types
        <operand types>
    Operators
        <operators>
    Properties
        <properties>
```

The following requirements have to be fulfilled by the syntax we want to develop:

- It has to have the ability to express both the static and dynamic properties of system properties where the static part should be left in the form as introduced by Davis and Yurchak (1984).
- The form of the syntax should be as simple and easy to understand as possible.
- The chosen names of the constructs of the syntax should be self-explanatory and suggest the intended meaning to the user.
- It has to provide a precise and unambiguous way to specify the system.

Another point to consider is that we want to follow certain accepted design principles, especially that of Information Hiding as it is done e.g. in the ADA package construct where there is a Package Interface (to provide the user of the package with all the information necessary to use the package and nothing else) and a Package Body (the implementation of the package).

Before we finalize the syntax for the dynamic specification of a system we need to say something about how the dynamic properties of static objects are described in

terms of places and transitions. Places reflect the conditions on the execution of the static functions, whereas transitions are used to describe changes in these conditions during the execution of the static functions. Some of these places and transitions are generic, i.e. they apply to any function. For example, a function is always **requested** by a place and becomes **activated** by an **activate**-transition. This does not prevent us from defining additional places and transitions when they are needed for a specification. We are going to use an uniform notation to indicate the connection between the statement of the dynamic properties and the static properties. Given an operator *storem : val, memaddr, state -> state* we will use internal places names that are preceded by **storem_** (e.g. storem_activated) and transition names that are followed by **_storem** (e.g. activate_storem). We also reserve standard notations for entry and exit places of subnets: entry places are always preceded by **req_** (e.g. req_storem for "request a store in memory") and exit places are always preceded by **avail_** (e.g. avail_storem for "result of store in memory is available). We reverse the order of attaching the name of the function because we want to emphasize that a subnet represents a transition that is specified in detail.

1. Places

There are three types of places we want to distinguish in our syntax:

- **internal places**, which names and definitions are only known and accessible within the net they are defined in; the purpose is to build the internal structure of the net

- **entry places**, which names and definition are also known and accessible to those nets which are declared to be extensions of this net; they provide an interface to invoke this net

- **exit places**, which names and definitions are known and accessible to those nets which are declared to be extensions of this net; they provide an interface to obtain the results from the invocation of this net

We have chosen the following syntax to describe places in our specification:

place_name(netlabel)[message_type].

A place_name is the distinct name of a place in the described net. In the case that a place is either an entry or exit place the rule applies that their names are known to those nets which are declared to be extensions of this net. If a net has multiple entry or exit places the parameter netlabel can be attached in parentheses to describe this fact where netlabel indicates a location in the system. As we have said a place is able to accommodate a certain kind of message so the type of the message the place can hold in brackets is part of the place description. The following description of places are legal (compare with the static specification for "fetchr" and "storem" in Chapter II):

- **storem_activated[val.memaddr.state]**; a place of the name "storem_activated" which can hold messages that consist of data of type val, memaddr and state

- **fetchr_avail[]**; a place of the "fetchm_avail" which can only hold empty messages

- **req_pushstk(netlabel)[val.stkaddr.state];** places of the name "req_pushstk" which are distinguished by the parameter "netlabel", all of them able to hold messages which contain data of types val, stkaddr and state.

2. Transitions

Our methodology defines transitions as the process of collecting a message from each connected input place and sending messages to every connected output place. So we want to state what kind of messages are received and transmitted. The following syntax is used to describe transitions:

transition_name:input_messages -> output_messages.

The transition_name is a distinct name for the transition in the net and is not known outside the net. Input and output messages are of the above form where multiple messages are separated by commas. The following examples are legal description of transitions:

- **perform_storem: [val.memaddr.state] -> [state];**
  the transition of the name "perform_storem" takes a message which contains data of type val, memaddr and state as input and outputs a message containing data of type state

- **finish_fetchr: [val],[] -> [val],[];**
  the transition of the name "finish_fetchr" takes two messages as input where one consists of data of type val and the other is an empty message and outputs again two messages of same type

Note that the description of transitions only declares them in terms of their capabilities to accept and to transmit certain kinds of messages and does not show any internal action of the transition. The reason for this is to present the transitions as building blocks of the net in form of a mapping function which is general enough to provide

information about its interface and nothing more. The internal actions are described as the properties of the net. This does not necessarily means that a transition is instantaneous, rather the complexity of the internal actions determine the duration of the transition. But every complex transition can be modeled as a net with entry and exit places such that the internal transitions become instantaneous.

3. Properties

Now that we have described the building blocks of a net we need to connect them in order to describe the intended timing of a system. The following form is chosen for the syntax of the properties of the net:

transition_name(place_names[message_data) =>

place_names[message_data]; This form shows how places and transitions are connected and how the transfer of message elements occurs between them. Here are some examples of legal property descriptions:

- **perform_fetchm(fetchm_activated[m,q]) => fetchm_completed[v];** the transition "perform_fetchm" occurs when there is a message in the place "fetchm_activated". The message is taken from the place in such a way that all message elements (memaddr m and state q) are available to the transition. Then the action of getting the memory contents is performed and the resulting value v is transmitted as a message to the place "fetchm_completed"

- **perform_storer(storer_activated[v,r,q]) => storer_completed[q1];** the transition "perform_storer" occurs when there is a message in the place "storer_activated" in such a way that the message is taken from the place in such a way that its message elements (value v, regaddr r and state q) are available to the transition. The action of storing the value v in

register r is performed and the new state q1 is transmitted as a message to the place "storer_completed"

Functionally, the internal actions performed by transitions follow the rules stated as static properties for the functions involved.

4. Initialization

In some kinds of nets we have an internal circuit of places in order to act as a synchronization mechanism (as illustrated in Figure 4.4 to provide mutual exclusion). They have to be initiated somehow i.e. a message has to be placed in at least one of these places since they are not provided with messages from outside the net. We going to describe this initialization by using the symbol "=>" used to indicate the placement of a message into a place. The following is an example of an initialization:

=> fetchm_avail[]; the place "fetchm_avail" is loaded with an empty message

The initialization of a place is a one-time action at the beginning of system start and provides the necessary conditions to get a process going. It can be viewed as establishing the initial state of a computer system when it is turned on.

V. THE ABSTRACT PROCESSOR TIMING SPECIFICATION

In this chapter we want to present some examples on how specific problems of timing in computer system can be modeled using the methodology in a top-down fashion. The examples resemble a variety of computer system timing problems to test the use of Petri Nets in specifying the timing properties. In Appendix B a complete specification of the static and dynamic properties of a reduced Abstract Processor is presented. We have chosen to use the Abstract Processor as the object to be specified in timing considerations because of the following reasons:

- to emphasis this work as the logical step following the work of specifying static properties of systems,
- by specifying a non-existent, abstracted processor we intentionally leave the issue of the actual implementation untouched since we stated that by whatever means the specification is implemented the processor will have the specified properties,
- to emphasize our intention of specifying **what** the user of a processor wants to achieve and **not how** it is implemented as compared to a traditional processor design approach that is dominated by engineering and implementation issues.

Also we want to show how well our methodology can deal with the special aspects of timing in computer systems. The special aspects we are concerned with are mutual exclusion, interrupt processing and concurrency. Despite the fact that the Abstract Processor is in its static part specified as a simple single processor during this work we have realized

that even there, a lot of potential concurrency can be detected.

Whenever we refer to actual implementation we do this with the intention to give one example of how the specification could be realized. Once again we emphasize that the methodology stated in this work is only concerned with what is available in a system and not with the how it is implemented. We want to remind the reader that the methodology developed here is intended to be general. That is, it can be equally applied to the specification of computer resources that may be implemented in hardware, software or firmware. With this in mind we have look at the timing specification not as a blueprint by which a system can be build directly but rather as formally stated requirements a system has to fullfil no matter what approach is chosen for the implementation .

A. ATOMIC NETS

One feature of the methodology is it forces the specifier to focus on the essential nature of the system components. When we consider these essential components as actions in a system that are not further divisible we can speak of atomic actions which we want to specify as atomic nets in our methodology. Such nets will use no other net in their specification and so can be considered as building blocks of a system. In general they represent the elementary actions in a system. The Abstract Processor consists of several such

elementary actions and to illustrate this idea, we will show how the actions of store operations and fetch operations can be described with atomic nets.

B. MODELING OF MEMORY AND REGISTER ACCESS TIMING

The first question we have to ask when we want to specify the access of memory or registers is what kind of information do we have to have available to perform an access and what information we obtain after the access has been made. In order to perform an access the address of the memory cell or register has to be available. Depending whether a store or a fetch has to be performed, we either have to provide a value for this process or we obtain a value from the process. Also, to indicate the current contents of the register or memory cell we have to indicate the process for accessing the state of the processor. In case of a store access we obtain a new state as the result of the process since the change of a memory cell or register also changes the state.

At this stage we have established the components of any process dealing with the access of memory or registers. We anticipate that accesses to memory and registers will be made from various places in the system, so we provide a netlabel for every entry and exit place. In terms of our specification methodology can now define the entry and exit places of the subnet:

- the fetching of the contents of a memory cell is requested by providing message which consists of the

memory address and the state to the following entry place: **req_fetchm(netlabel)[memaddr.state];**

- We obtain as a result a corresponding message containing the value from the exit place: **avail_fetchm(netlabel)[val];**

- similarly we state the entry and exit places for fetching the contents of a register: **req_fetchr(netlabel)[regaddr.state];** as the entry place and **avail_fetchr(netlabel)[val];** as the exit place

- the storing of a value into a memory cell is requested by providing a message which consists of the value to be stored, the memory address and the state to the following entry place: **req_storem(netlabel)[val.memaddr.state];**

- We receive as a result a corresponding new state from the exit place: **avail_storem(netlabel)[state];**

- similarly we define the entry and exit places for storing values into registers; **req_storer(netlabel)[val.regaddr.state];** as the entry place and **avail_storer(netlabel)[state];** as the exit place.

To show the versatility of the proposed methodology we will specify memory and register access differently: we are going to specify memory access in a way that allows only one access to one memory cell at a time (an implementation for this method might be a single memory unit allowing only one access at a time). On the other hand we might want to able to access registers in parallel. These requirements determine the internal structure of resulting specification.

Considering the above requirements on how we have to specify the system we realize that we have to construct the specification to deal with memory accesses and accesses to each register separately.

As the next step we are going to specify the memory accesses. Since we know from our requirements that only one access at a time is allowed to the memory we have to provide a mutual exclusion mechanism in our specification that makes sure that the memory is not accessed by more than one request at a time.

From Figure 5.1 we see that from any system location indicated by a netlabel a message in an entry place to request a memory access can only trigger a transition to activate the process when the process is available. Since there is only one empty message to indicate the availability of the net only one request can be honored at a time. The availability is restored when the access is completed. Also we see that the internal places "processed_for" serve as traffic signs to direct the results to the appropriate exit places. Drawing the picture of the net can be helpful to the process of specifying net just as flow charts can be helpful in programming tasks, but our intention is primarily to state a formal specification. The following is the dynamic part of the memory access specification expressed in precise syntax:

```
entry places
    req_fetchm(netlabel)[memaddr.state];
    req_storem(netlabel)[val.memaddr.state];

exit places
    avail_fetchm(netlabel)[val];
    avail_storem(netlabel)[state];

internal places
    access_avail[];
    fetchm_for(netlabel)[];
    fetchm_activated[memaddr.state];
```

```
    fetchm_completed[val];
    storem_for(netlabel)[];
    storem_activated[val.memaddr.state];
    storem_completed[state];

initial state
    => access_avail[];

transitions
    act_fetchm: [memaddr.state],[] -> [memaddr.state],[];
    perform_fetchm: [memaddr.state] -> [val];
    finish_fetchm: [val],[] -> [val],[];
    act_storem: [val.memaddr.state],[] ->
    [val.memaddr.state],[];
    perform_storem: [val.memaddr.state] -> [state];
    finish_storem: [state],[] -> [state],[];

properties
    act_fetchm(req_fetchm(l1)[m.q], access_avail[]) =>
        fetchm_for(l1)[],fetchm_activated[m.q];
    perform_fetchm(fetchm_activated[m.q]) =>
        fetchm_completed[v];
    finish_fetchm(fetch_completed[v], fetchm_for(l1)[]) =>
        avail_fetchm(l1)[v], access_avail[];
    act_storem(req_storem(l1)[v.m.q], access_avail[]) =>
        storem_for(l1)[], storem_activated[v.m.q];
    perform_storem(storem_activated[v.m.q]) =>
        storem_completed[q1];
    finish_storem(storem_completed[q1], storem_for(l1)[]) =>
        avail_storem(l1)[q1], access_avail[];
```

When we look at the requirements of the register accesses we see that the above design for memory accesses is not suitable for register access if we want to allow for concurrent access to registers. So we have to find a way to express the properties of register accesses. Looking closely again at the requirements we realize that each register has the same access policies as the whole memory we specified before. This leads us to the fact that every register has to have its own specification. For the sake of simplicity let us say that our system has three registers with register addresses 1,2 and 3. We could now define three different nets

each for the access of a certain register. There is one problem though: whenever a register access is requested from somewhere in the system, the proper net for the register to be accessed has to be addressed. So we want to have the decision about which register net is meant centralized in one place in the system. Therefore we employ some decision

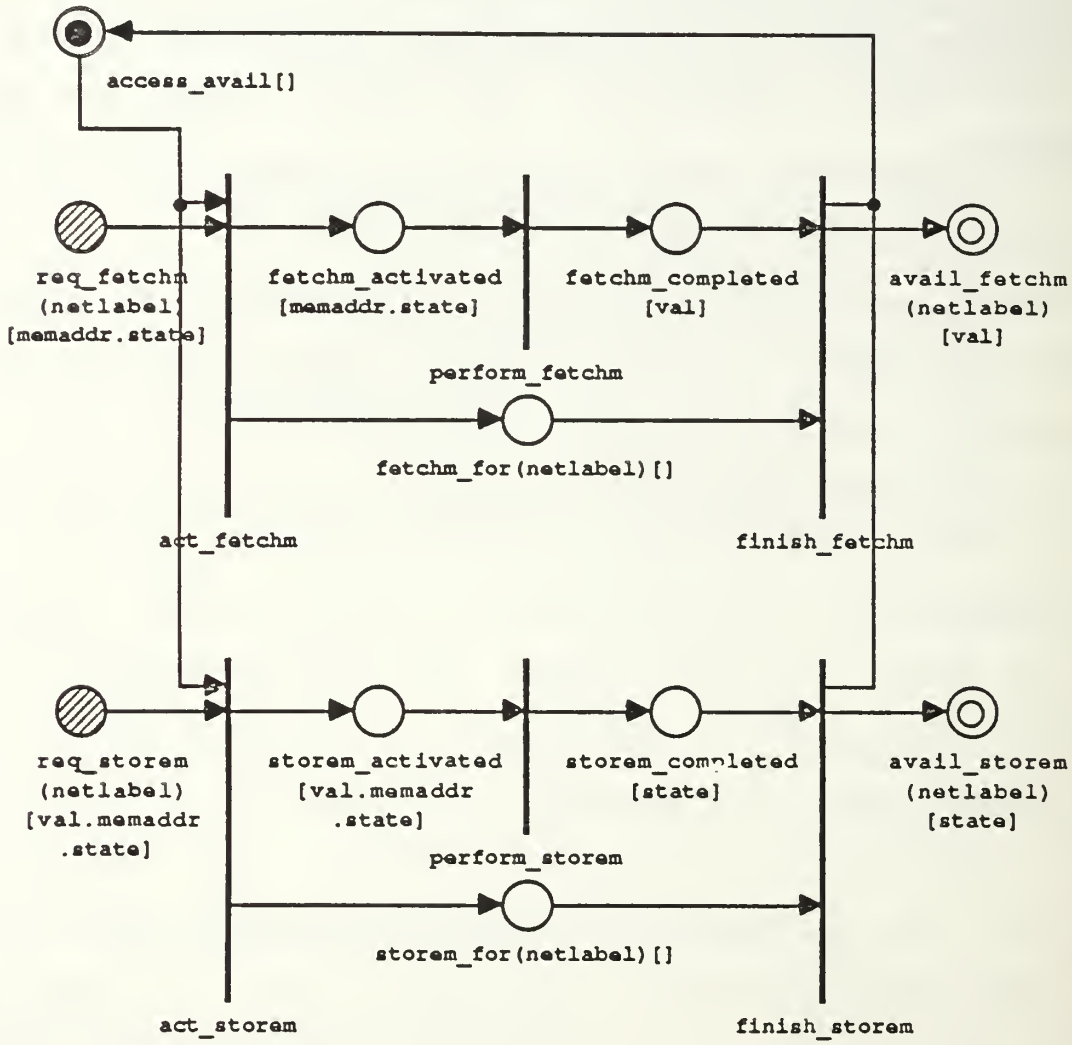

Figure 5.1: Petri Net Graph for Memory Access

mechanism to select the right register access net. In this case the graph drawn out in Figure 5.2 of the net might confuse more than it helps. We try to specify the register

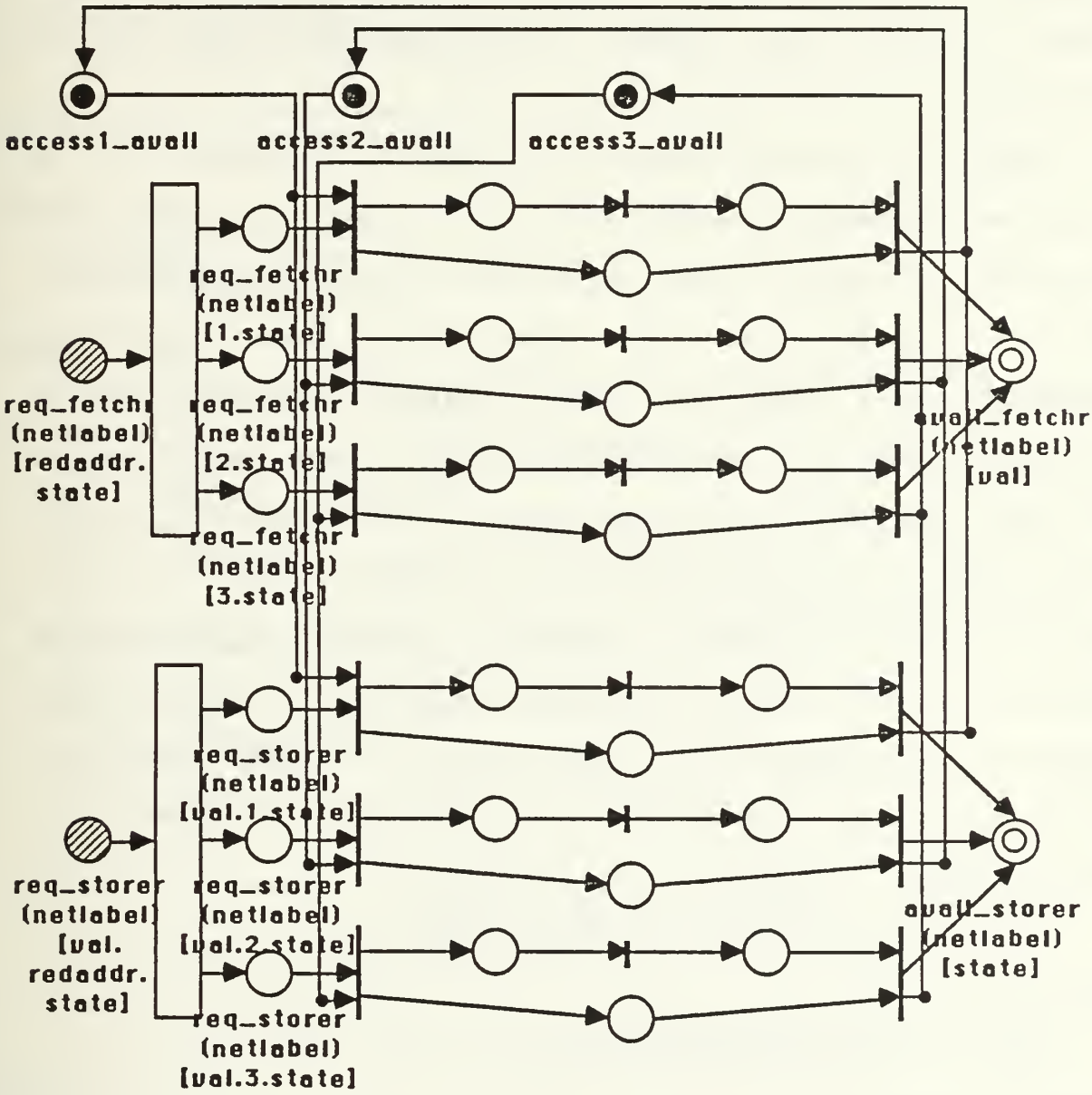

Figure 5.2: Petri Net Graph for Register Access

access just by direct reasoning. Still, the graph in Figure 5.2 illustrates that, despite the fact that each register can be accessed independently from the other registers, only oneaccess to a register is allowed at a time because of the separate "access_avail" places for each register. These places prohibit any access to a register until it is available.

Again, we can anticipate that access to registers will be requested by several locations in the system. So we can state the entry and exit places as we have in the memory example. Next we already found out that there three independent register access nets and we can use again this structure for internal places and transitions we used in the memory access net. But how do we state that the net for register 1 is used when there is an access request for register 1 and only this net? The indication that a certain register is going to be accessed is the register address contained in the request message. Now instead of a name of the register address we state the actual value of it when we specify the properties:

```
entry places
    req_fetchr(netlabel)[regaddr.state];
    req_storer(netlabel)[val.regaddr.state];

exit places
    avail_fetchr(netlabel)[val];
    avail_storer(netlabel)[state];

internal places
    access1_avail[];
    fetchr1_activated[regaddr.state];
    fetchr1_completed[val];
    storer1_activated[val.regaddr.state];
    storer1_completed[state];
```

```
    access2_avail[];
    fetchr2_activated[regaddr.state];
    fetchr2_completed[val];
    storer2_activated[val.regaddr.state];
    storer2_completed[state];

    access3_avail[];
    fetchr3_activated[regaddr.state];
    fetchr3_completed[val];
    storer3_activated[val.regaddr.state];
    storer3_completed[state];

    fetchr_for(netlabel)[];
    storer_for(netlabel)[];

initial state
    => access1_avail[];
    => access2_avail[];
    => access3_avail[];

transitions
    act_fetchr1: [regaddr.state],[] -> [regaddr.state],[];
    perform_fetchr1: [regaddr.state] -> [val];
    finish_fetchr1: [val],[] -> [val],[];
    act_storer1: [val.regaddr.state],[] ->
    [val.regaddr.state],[];
    perform_storer1: [val.regaddr.state] -> [state];
    finish_storer1: [state],[] -> [state],[];

    act_fetchr2: [regaddr.state],[] -> [regaddr.state],[];
    perform_fetchr2: [regaddr.state] -> [val];
    finish_fetchr2: [val],[] -> [val],[];
    act_storer2: [val.regaddr.state],[] ->
    [val.regaddr.state],[];
    perform_storer2: [val.regaddr.state] -> [state];
    finish_storer2: [state],[] -> [state],[];

    act_fetchr3: [regaddr.state],[] -> [regaddr.state],[];
    perform_fetchr3: [regaddr.state] -> [val];
    finish_fetchr3: [val],[] -> [val],[];
    act_storer3: [val.regaddr.state],[] ->
    [val.regaddr.state],[];
    perform_storer3: [val.regaddr.state] -> [state];
    finish_storer3: [state],[] -> [state],[];

properties
    act_fetchr1(req_fetchr(l1)[1.q], access1_avail[]) =>
        fetchr_for(l1)[],fetchr1_activated[1.q];
    perform_fetchr1(fetchr1_activated[1.q]) =>
        fetchr1_completed[v];
    finish_fetchr1(fetch1_completed[v], fetchr_for(l1)[]) =>
        avail_fetchr(l1)[v], access1_avail[];
```

```
act_storer1(req_storer(l1)[v.1.q], access1_avail[]) =>
    storer_for(l1)[], storer1_activated[v.1.q];
perform_storer1(storer1_activated[v.1.q]) =>
    storer1_completed[q1];
finish_storer1(storer1_completed[q1], storer_for(l1)[])
    => avail_storer(l1)[q1], access1_avail[];

act_fetchr2(req_fetchr(l1)[2.q], access2_avail[]) =>
    fetchr_for(l1)[],fetchr2_activated[2.q];
perform_fetchr2(fetchr2_activated[2.q]) =>
    fetchr2_completed[v];
finish_fetchr2(fetch2_completed[v], fetchr_for(l1)[]) =>
    avail_fetchr(l1)[v], access2_avail[];
act_storer2(req_storer(l1)[v.2.q], access2_avail[]) =>
    storer_for(l1)[], storer2_activated[v.2.q];
perform_storer2(storer2_activated[v.2.q]) =>
    storer2_completed[q1];
finish_storer2(storer2_completed[q1], storer_for(l1)[])
    => avail_storer(l1)[q1], access2_avail[];

act_fetchr3(req_fetchr(l1)[3.q], access3_avail[]) =>
    fetchr_for(l1)[],fetchr3_activated[3.q];
perform_fetchr3(fetchr3_activated[3.q]) =>
    fetchr3_completed[v];
finish_fetchr3(fetch3_completed[v], fetchr_for(l1)[]) =>
    avail_fetchr(l1)[v], access3_avail[];
act_storer3(req_storer(l1)[v.3.q], access3_avail[]) =>
    storer_for(l1)[], storer3_activated[v.3.q];
perform_storer3(storer3_activated[v.3.q]) =>
    storer3_completed[q1];
finish_storer3(storer3_completed[q1], storer_for(l1)[])
    => avail_storer(l1)[q1], access3_avail[];
```

We see that even specifications of simple resources become large and complex and the drawing the net is even more complex. Here we realize the real benefit of the introduction of subnets: once these subnets are specified we can use their properties everywhere in our system simply by stating the entry and exit places of the subnets in the net we want to specify. Those nets using subnets are actually extensions of the subnets. The next section will illustrate these facts in detail.

C. MODELING OF INSTRUCTION FETCH AND EXECUTION TIMING

From the static specification of the Abstract Processor we see that there are two operands "prog" and "exq" which are responsible for the process of executing programs. The process is kept going by corecursive calls between those two operators.

Even though those two processes are very closely connected by corecursive calls we want to consider them separately and start with specifying the "exq" process.

The "exq" process needs information about the instruction to be executed, the current memory address, and the state of the processor. After the process has finished it returns the new state. This determines the contents of the messages the entry and exit places of this net have to accommodate. Still we have to decide what kind of execution unit we want to specify. We want to specify that only one execution unit can be performed at a time. This means that we have to provide mutual exclusion for the use of this net. We can do this the same way as we provided for memory accesses. We can accomplish that by providing a distinct entry place and exit place indicated by netlabels. Now we are in a position to specify the entry and exit places:

- **req_exq(netlabel)[instr.memaddr.state];** as the entry place
- **avail_exq(netlabel)[memaddr.state];** as the exit place

At this point we have to look closely at the actions an execution on an instruction has to accomplish: retrieval of

information from the instruction, about register and memory addresses of operands, and about the operation to be performed, accesses to the operands, application of the operator, and calculation of the address of the next instruction to execute. We see now that the previous specifications for memory and register accesses will come in handy when have to specify these accesses in the specification. Also we assume for this specification that there are nets for the retrieval of operands and operators from the instruction, the application of operands and calculation of the next memory address.

The next issue we have to address is the fact that different instructions have to be executed differently. Here a decision mechanism similar to the one we introduced to access a specific register can help us to make the specification structured and understandable. The mechanism we introduce here has to recognize from the instruction part of the message which execution is requested and has to direct the path within the specification to the appropriate part of the specification. In the formal specification we indicate this explicitly by the contents of the instruction part of the request message.

In the following we show some representative examples of the execution specifications of different instructions. Their graphs are depicted in Figures 5.4 and 5.5 and illustrate

clearly the simplification obtained by the use of predefined subnets.

```
entry places
    req_exq(netlabel)[instr.memaddr.state];

exit places
    avail_exq(netlabel)[memaddr.state];

internal places
    exq_avail[];
    exq_for(netlabel)[];

    exq_monad_activated[state];
    exq_monad_fetch[state];
    exq_monad_apply[state];
    exq_monad_store[];

    exq_mov_r_r_activated[state];
    exq_mov_r_r_perform[state];
    exq_mov_r_r_store[];

initial state
    => exq_avail[];

transitions
    activate_exq_monad: [instr.memaddr.state],[] ->
        [state],[instr],[instr],[instr],[memaddr];
    start_exq_monad: [state],[regaddr], ->
        [state],[regaddr.state];
    apply_exq_monad: [state],[operator],[val] ->
        [state],[operator.val];
    store_exq_monad: [state],[val],[regaddr] ->
        [],[val.regaddr.state];
    finish_exq_monad: [],[state],[memaddr] ->
        [memaddr.state];

    activate_exq_mov_r_r: [instr.memaddr.state],[] ->
        [state],[instr],[instr],[memaddr];
    start_exq_mov_r_r: [start],[regaddr] ->
        [state],[regaddr.state];
    store_exq_mov_r_r: [state],[regaddr],[val] ->
        [],[val.regaddr.state];
    finish_exq_mov_r_r: [],[state],[memaddr] ->
        [memaddr.state];

properties
    activate_exq_monad(exq_avail[],
        req_exq(l)[monad(o.r1.r2).m.q]) =>
        exq_for(l)[], exq_monad_activated[q],
        req_operator(l1)[monad(o.r1.r2)],
```

```
        req_operand1(12)[monad(o.r1.r2)],
        req_operand2(13)[monad(o.r1.r2)],
        req_nextmemaddr(14)[m];
    start_exq_monad(exq_monad_activated[q],
        avail_operand1(11)[r1]) -> exq_monad_fetch[q],
        req_fetchr(15)[r1.q];
    apply_exq_monad(exq_monad_fetch[q], avail_fetchr(15)[v],
        avail_operator(11)[o]) => exq_monad_apply[q],
        req_apply_mop(16)[o.v];
    store_exq_monad(exq_monad_apply[q],
        avail_apply_mop(16)[v1], avail_operand2(13)[r2]) =>
        exq_monad_store[], req_storer(17)[v1.r2.q];
    finish_exq_monad(exq_monad_store[], avail_storer(17)[q1],
        avail_nextmemaddr(14)[m1]) => avail_exq(1)[m1.q1];

    activate_exq_mov_r_r(exq_avail[],
        req_exq(1)[mov_r_r(r1,r2).m.q]) =>
        exq_mov_r_r_activated[q],
        req_operand1(11)[mov_r_r(r1,r2)],
        req_operand2(12)[mov_r_r(r1,r2)],
        req_nextmemaddr(13)[m];
    start_exq_mov_r_r(exq_mov_r_r_activated[q],
        avail_operand1(11)[r1]) =>
        exq_mov_r_r_perform[q], req_fetchr(14)[r1.q];
    store_exq_mov_r_r(exq_mov_r_r_perform[q],
        avail_fetchr(14)[v], avail_operand2(12)[r2]) =>
        exq_mov_r_r_store[], req_storer[v.r2.q];
    finish_exq_mov_r_r(exq_mov_r_r_store[], avail_storer[q1],
        avail_nextmemaddr(13)[m1]) =>
        avail_exq(1)[m1.q1];
```

The above two examples show the specification of the execution of a monadic instruction and of a move instruction. We have used the previously specified register access ("fetchr" and "storer") to fetch the contents of a register and to store a value into a register. The way we used them was that we stated their entry and exit places at the appropriate locations in the description of the properties of our execution specification. In the same way we invoked the specifications of "nextmemaddr", "apply_mop", "operand1", "operand2" and "operator" which for this example we assumed to be defined .

We want to emphasize at this point that the stated properties of the given examples are not the only way the execution could be specified: we are following the philosophy that as soon as the information is available, the possible requests are made based on the information. In a real specification other considerations may have priority, but our intention is to show how this can be accomplished using the methodology.

The next part of the specification is to state the "prog" part and to connect it with "exq" in a way that illustrates the corecursiveness of their interconnection. Since we have already modeled the "exq" part as a process taking an instruction, a memory address and the state, and provides a new memory address and a new state, we want this as a subnet in our "prog" specification. When we look again at the static specification of "prog" of the Abstract Processor we realize that this process needs a memory address and the state to get started, i.e. the same information as "exq" provides as output. This fact leads to the idea of a loop in requesting "prog": when "prog" has performed its initial tasks and has invoked "exq" it is in the situation of requesting itself again as soon as the result of "exq" is obtained. This idea will work nicely once the process is started, but how do we get this process running? Here we can claim that the initial request has to come from the outside world (imagine a possible implementation as an on-switch at the machine which

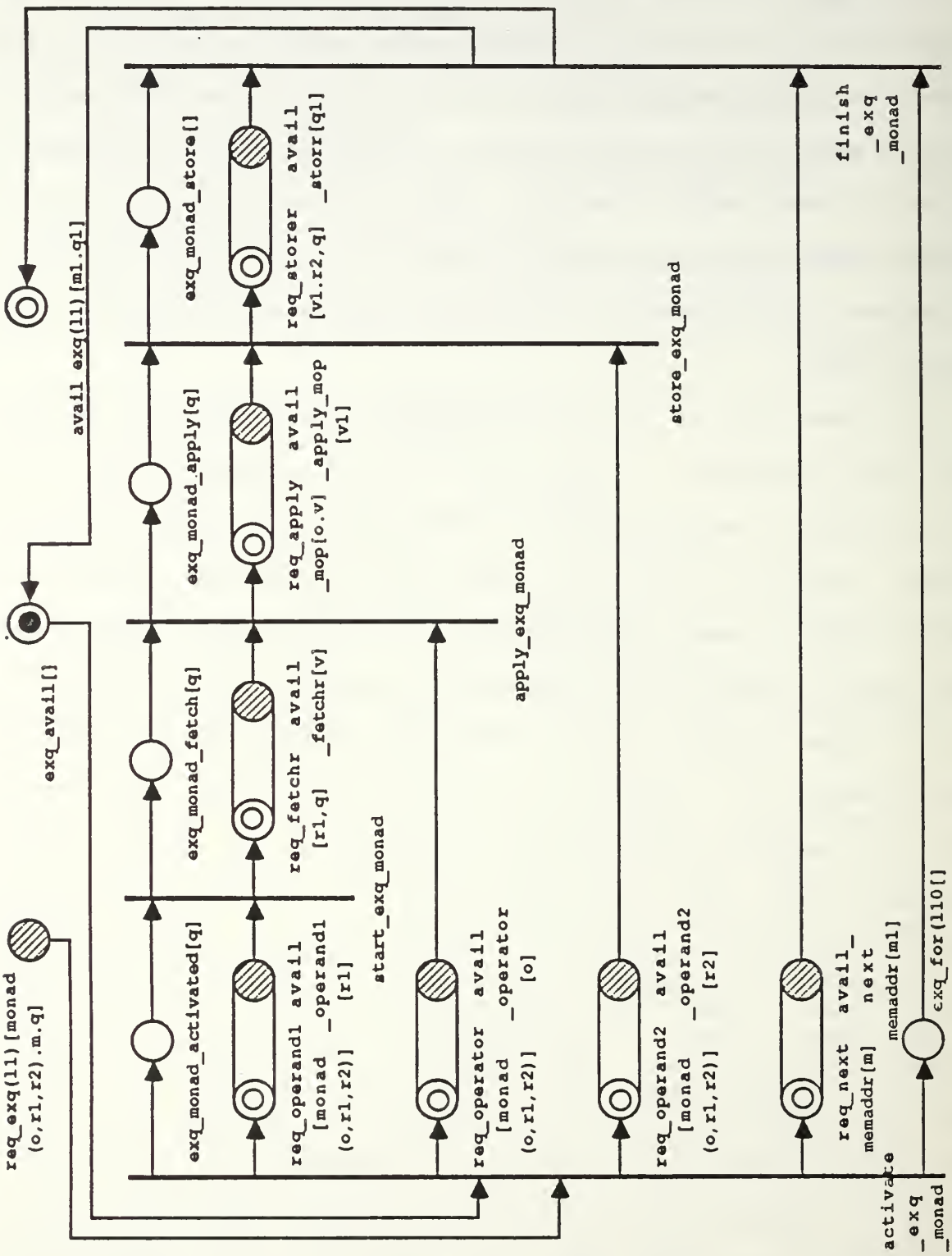

Figure 5.3: Partial Petri Net Graph of "exq_monad"

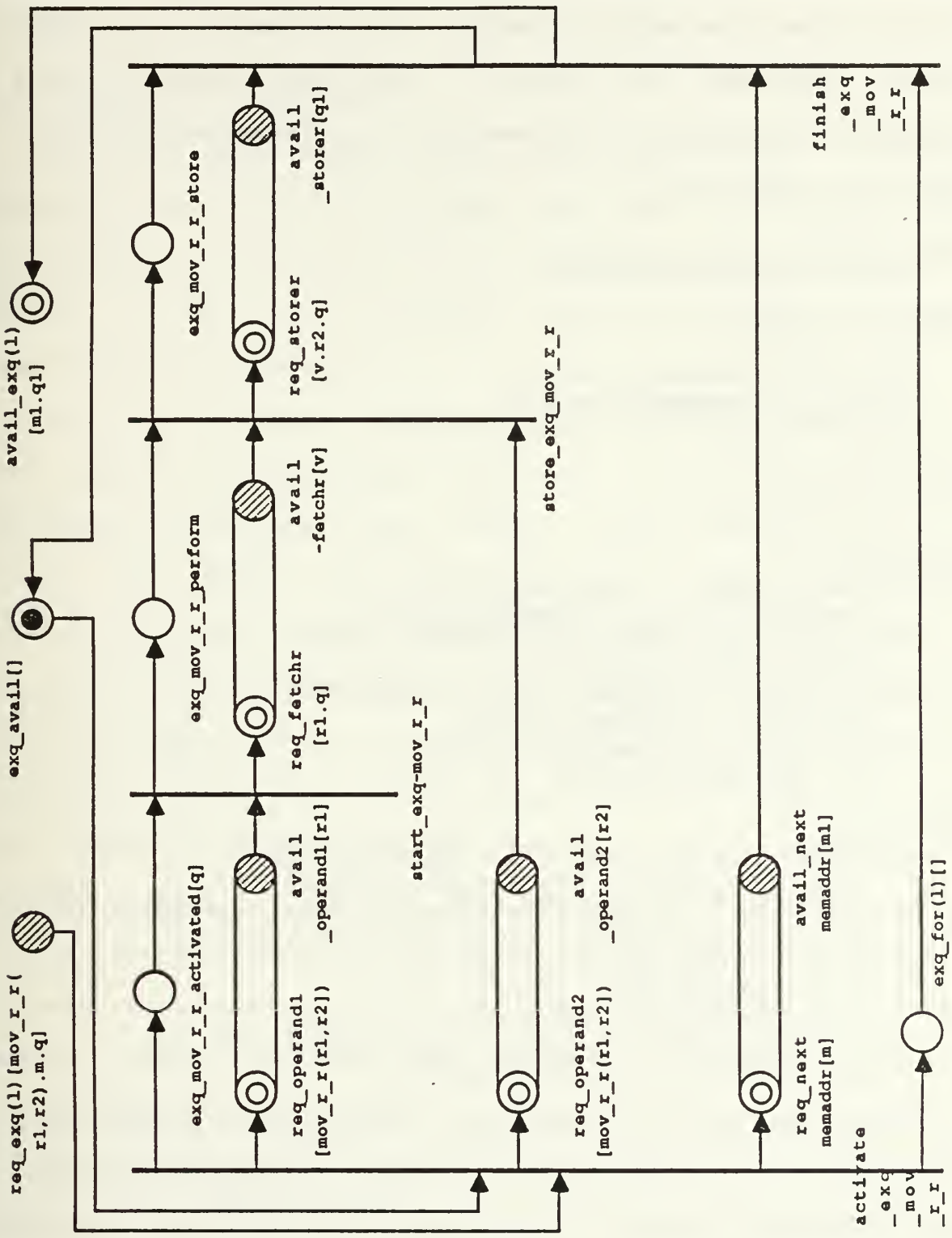

Figure 5.4: Partial Petri Net Graph of "exq_mov_r_r"

resets the state and sets the program counter to a predefined initial value). We want to specify this process as a single control unit and that this is the major process in the system. The following is a possible specification of "prog" (see also Figure 5.5):

```
entry places
    req_prog[memaddr.state];

internal places
    prog_avail[];
    prog_fetch[memaddr.state];
    prog_instr[memaddr.state];
    prog_perform[]

initial state
    => prog_avail[];

transitions
    activate_prog: [],[memaddr.state] ->
        [memaddr.state],[memaddr.state];
    get_instr_prog: [memaddr.state],[val] ->
        [memaddr.state],[val];
    perform_prog: [memaddr.state],[instr] ->
        [],[instr.memaddr.state];
    finish_prog: [],[memaddr.state] ->
        [],[memaddr.state];

properties
    activate_prog(prog_avail[], req_prog[m.q]) =>
        prog_fetch[m.q], req_fetchm(l1)[m.q];
    get_instr_prog(prog_activated[m.q], avail_fetchm(l1)[v])
        => prog_instr[m.q], req_atomofinstr(l2)[v];
    perform_prog(prog_instr[m.q],
        avail_atomofinstr(l2)[i]) =>
        prog_perform[], req_exq(l3)[i.m.q];
    finish_prog(prog_perform[], avail_exq[m1.q1]) =>
        prog_avail[], req_prog[m1.q1];
```

The declaration of "req_prog" as an entry place models our previous stated connection to the outside world. Note that this specification in the form presented could not be used as a subnet since no exit place has been declared. The

following Figure 5.5 shows how this specification is drawn as a net.

D. MODELING OF INTERRUPTS

The above specifications of "exq" and "prog" and their interconnection in their current form does not provide for any recognition and processing of interrupts. This chapter is to show one way how interrupts could be handled using our specification methodology.

As always the first step is to state the requirements for the process of interrupt handling. We want to look at interrupts as a signal which comes either from outside or inside the system on which the current running program is interrupted and a handler program at a predefined location is started. After the completion of the handler the execution of the interrupted program is resumed, therefore we need to save the memory address of the interrupted program. Also we want the invocation of the interrupt handler only to happen at a defined state, i.e. between the completion of the execution of one instruction and the fetching of the next instruction. Another requirement is that the interrupt handler acts on a single interrupt only one time, so that a following interrupt signal can be interpreted as another interrupt. In the following specification we assume that there is a dedicated stack in the system on which the current memory address of

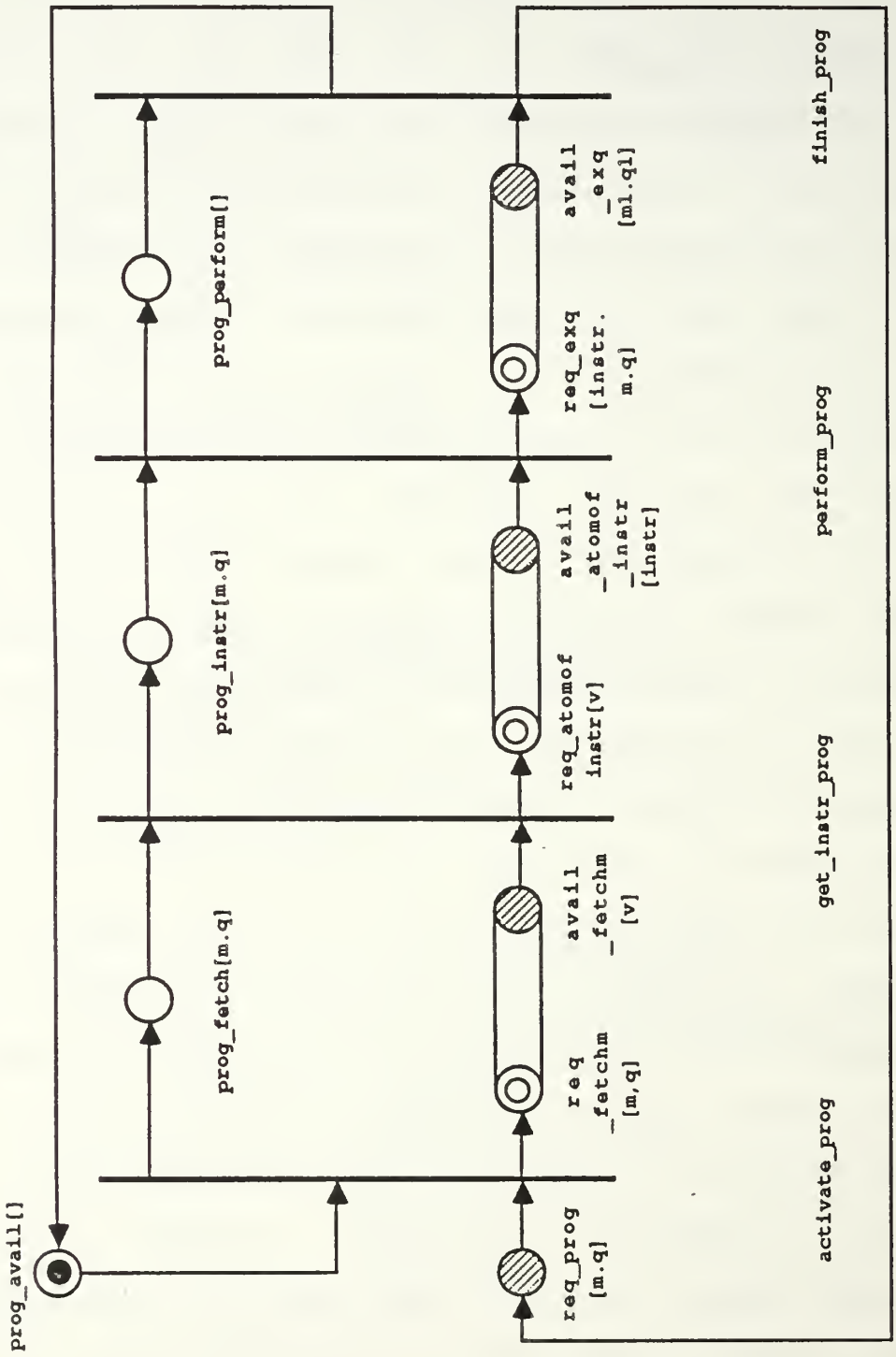

Figure 5.5: Graph of "prog" without Interrupt Handling

the running program can be saved. Also, for simplicity reasons we allow only one type of interrupt, i.e. the handler has to determine the source of the interrupt by software. The following is one way to specify the above requirements (also see Figure 5.6):

```
entry places
    req_prog[memaddr.state];

internal places
    prog_avail[];
    prog_fetch[memaddr.state];
    prog_instr[memaddr.state];
    prog_perform[]
    prog_check[memaddr.state];
    prog_save[];

initial state
    => prog_avail[];

transitions
    activate_prog: [],[memaddr.state] ->
        [memaddr.state],[memaddr.state];
    get_instr_prog: [memaddr.state],[val] ->
        [memaddr.state],[val];
    perform_prog: [memaddr.state],[instr] ->
        [],[instr.memaddr.state];
    finish_prog: [],[memaddr.state] ->
        [memaddr.state],[];

    normal_prog: [memaddr.state],[] ->
        [],[memaddr.state];
    itrpt_prog:[memaddr,state],[] ->
        [],[instr.memaddr.state];
    fin_int_prog: [],[memaddr.state] ->
        [],[memaddr.state];

properties
    activate_prog(prog_avail[], req_prog[m.q]) =>
        prog_fetch[m.q], req_fetchm(l1)[memaddr.state];
    get_instr_prog(prog_activated[m.q], avail_fetchm(l1)[v])
        => prog_instr[m.q], req_atomofinstr(l2)[v];
    perform_prog(prog_instr[m.q],
        avail_atomofinstr(l2)[i]) =>
        prog_perform[], req_exq(l3)[i.m.q];
    finish_prog(prog_perform[], avail_exq[m1.q1]) =>
        prog_check[m1.q1], req_check[];
```

```
normal_prog(prog_check[m1.q1], avail_normal[]) =>
    prog_avail[], req_prog[m1.q1]
itrpt_prog(prog_check[m1.q1], avail_intrpt[]) =>
    prog_save[], req_exq[jsr(int_addr,sys_stk).m1.q1];
fin_int_prog(prog_save[], avail_exq[m2.q2]) =>
    prog_avail[], req_prog[m2.q2];
```

The above interrupt-sensitive specification of "prog" is very similar to the former specification of "prog" which did not provide for interrupt handling (compare Figure 5.5 and Figure 5.6). The major difference is that an "external agent" is invoked as soon as the execution of the instruction is completed. This Agent sends a message to one of its output places to show that either an interrupt is present or not. This construct ensures two properties: first, the External Agent is responsible for clearing the interrupt signal after it has recognized it so that an interrupt is only honored once; second, the presence of an interrupt has priority over the normal way of execution of a program. Once an interrupt has been detected by the Agent and its appropriate output place has been provided with a message the "prog" process can continue and it will place the current memory address on a specified system stack by executing a "push" instruction and will start a new cycle at the system interrupt handler address. Since the interrupt handler is by itself a loaded user program is responsible for saving the necessary register contents and for restoring those registers and the saved memory address from the system stack when it finishes.

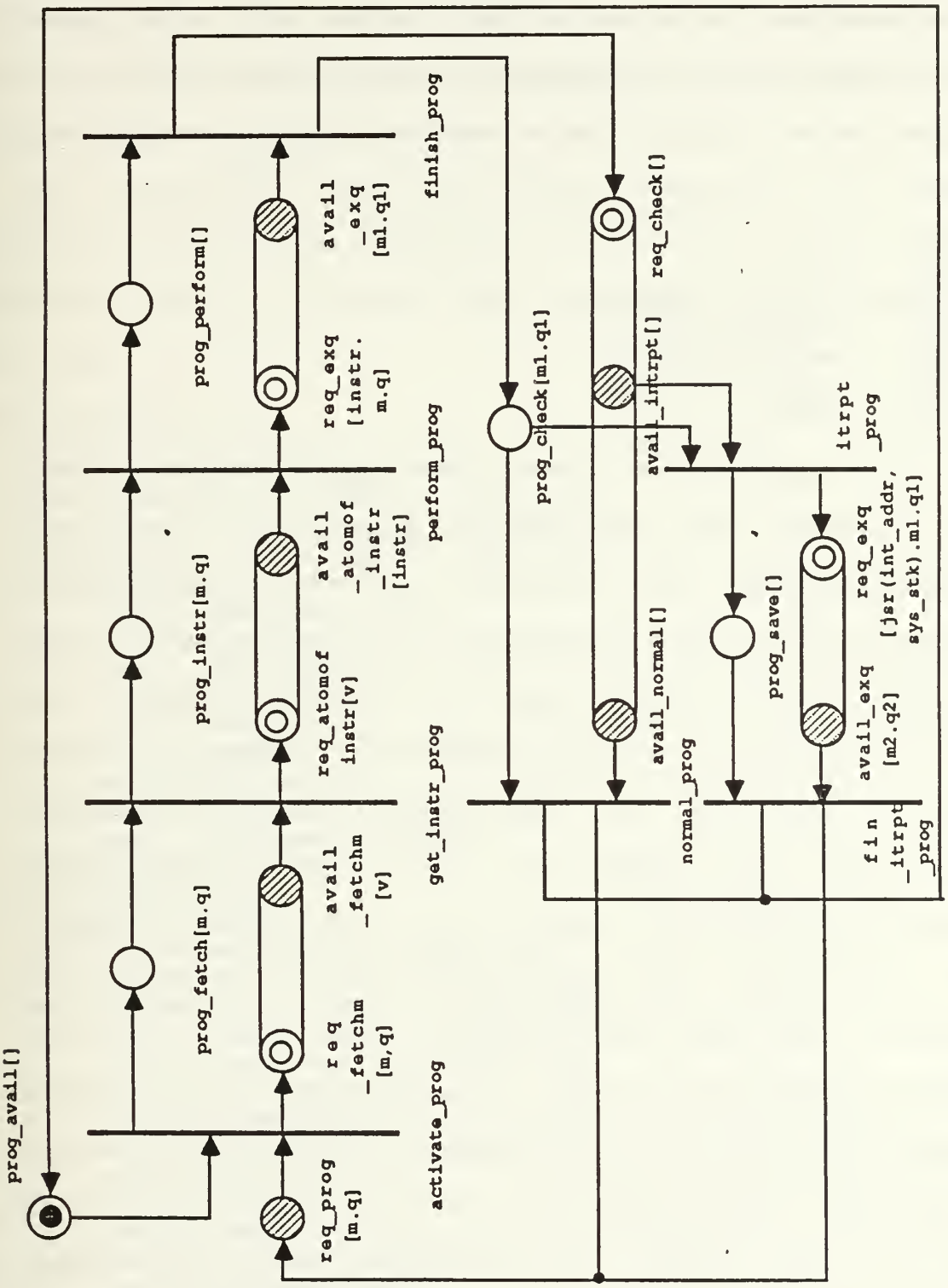

Figure 5.6: Graph of "prog" with Interrupt Handling

E. EXECUTION OF PROGRAMS

The question to be asked now is: how will the proposed specification methodology show the timing of the execution of programs? As an example, let us consider the following simple program:

```
1000:   1
1001:   2
2000:   MOV_M_R 1000,r1
2001:   MOV_M_R 1001,r2
2002:   ADD r1,r2,r3
2003:   MOV_R_M r3,1002
2004:   STOP
```

The above program retrieves the values "1" and "2" from memory addresses 1000 and 1002 into registers "r1" and "r2", adds them together, with the result in "r3", and than moves this result into memory address 1002. At the top level of the specification there is the "prog" net. With the start of this program it receives the "req_prog[2000.q1]"[4] message from the outside. It retrieves the instruction contained in memory address 2000 and invokes the "exq" net with the message "req_exq[instr.2000.q1]". Inside the "exq" net the "external agent" determines the appropriate net to process the instruction, here the "mov_m_r" net, and enters this net. After the completion of "exq", "prog" finishes with a message "req_prog[2001.q2]" to itself where the "2001" and q2 were obtained from the execution of "exq". At this point "prog" starts all over again and finishes with the message "req_prog[2002.q3]". This repeats until the "stop"

---

[4]Netlabels are omitted for simplicity

instruction is executed and results in termination of the process. Inside the invocations of the "exq" net there are several uses of the nets to access memory and registers as well as to process "nextmemaddr".

The following shows the major messages with their included data that are exchanged during the course of the execution of the program (not necessarily in the order as they might occur):

```
req_prog[2000,q1]
    req_fetchm[2000,q1]
    avail_fetchm["MOV_M_R 1000,r1"]
    req_atomofinstr["MOV_M_R 1000,r1"]
    avail_atomofinstr[mov_m_r(1000,r1)]
    req_exq[mov_m_r(1000,r1).2000.q1]
        req_operand1[mov_m_r(1000,r1)]
        avail_operand2[1000]
        req_operand2[mov_m_r(1000,r1)]
        avail_operand2[r1]
        req_fetchm[1000.q1]
        avail_fetchm[1]
        req_storer[1.r1.q1]
        avail_storer[q2]
        req_nextmemaddr[2000]
        avail_nextmemaddr[2001]
    avail_exq[2001.q2]
req_prog[2001,q2]
    req_fetchm[2001,q2]
    avail_fetchm["MOV_M_R 1001,r2"]
    req_atomofinstr["MOV_M_R 1001,r2"]
    avail_atomofinstr[mov_m_r(1001,r2)]
    req_exq[mov_m_r(1001,r2).2001.q2]
        req_operand1[mov_m_r(1001,r2)]
        avail_operand2[1001]
        req_operand2[mov_m_r(1001,r2)]
        avail_operand2[r2]
        req_fetchm[1001.q2]
        avail_fetchm[2]
        req_storer[2.r2.q2]
        avail_storer[q3]
        req_nextmemaddr[2001]
        avail_nextmemaddr[2002]
    avail_exq[2002.q3]
req_prog[2002.q3]
    req_fetchm[2002,q3]
```

```
    avail_fetchm["ADD r1,r2,r3"]
    req_atomofinstr["ADD r1,r2,r3"]
    avail_atomofinstr[dyadr(add,r1,r2,r3)]
    req_exq[dyad(add,r1,r2,r3).2002.q3]
        req_operand1[dyad(add,r1,r2,r3)]
        avail_operand1[r1]
        req_operand2[dyad(add,r1,r2,r3)]
        avail_operand1[r2]
        req_operand3[dyad(add,r1,r2,r3)]
        avail_operand3[r1]
        req_operator[dyad(add,r1,r2,r3)]
        avail_operator[add]
        req_fetchr[r1.q3]
        avail_fetchr[1]
        req_fetchr[r2.q3]
        avail_fetchr[2]
        req_applydop[add.1.2]
        avail_apply_dop[3]
        req_storer[3.r3.q3]
        avail_storer[q4]
        req_nextmemaddr[2002]
        avail_nextmemaddr[2003]
    avail_exq[2003.q4]
req_prog[2003,q4]
    req_fetchm[2003,q4]
    avail_fetchm["MOV_R_M r3,1003"]
    req_atomofinstr["MOV_r_m r3,1003"]
    avail_atomofinstr[mov_r_m(r3,1003)]
    req_exq[mov_r_m(r3,1003).2003.q4]
        req_operand1[mov_r_m(r3,1003)]
        avail_operand2[r3]
        req_operand2[mov_r_m(r3,1003)]
        avail_operand2[1003]
        req_fetchr[r3.q4]
        avail_fetchm[3]
        req_storer[3.1003.q4]
        avail_storer[q5]
        req_nextmemaddr[2003]
        avail_nextmemaddr[2004]
    avail_exq[2004.q5]
req_prog[2004.q5]
    req_fetchm[2004,q5]
    avail_fetchm["STOP"]
    req_atomofinstr["STOP"]
    avail_atomofinstr[stop]
    req_exq[stop.2003.q4]
    avail_exq[.q4]
```

With the appropriate tools to track certain messages and their contents one is able to take snapshots during the course of the execution to determine the timing within the execution.

VI. SUMMARY AND CONCLUSION

The intention of this work has been to present a methodology to specify timing in computer systems with strong emphasis on the issues of portability and reusability. The work has been also influenced by the goal to pursue a practical viewpoint for dealing with computer resources. In describing the essential properties of timing between abstracted computer resources, it has been possible to state the required timing in a system in an exact and rigorous way. As a first result of this work it has been pointed out that the attempt to specify the time behavior of a computer system without having a formal specification of the static behavior of the system will lead to inconsistencies and errors. We view the timing specification as an extension of the static specification (the system functionally) to a complete specification (the system functionally and dynamically).

A. ADVANTAGES

This methodology is based on Petri Nets and their underlying theory. Since Petri Nets are an accepted and commonly used tool in a variety of applications one familiar with Petri Nets will have almost no problems understanding the methodology. During the course of this research a number of distinct advantages have been recognized:

1. Ability to State Asynchronuous Timing

Asynchronuous timing in a system is the most common timing method in any computer system, be it the reaction on the completion of some task or dealing with an external interrupt. The examples presented during this work show very clearly that every event displays asynchronuous timing since it reacts on certain holding conditions whenever they might be true. By stating events and their connected places we can describe the asynchronuous timing easily.

2. Ability to Show Dataflow in a System

The combined consideration of timing and dataflow in a system has been accomplished by changing the original token meaning of Petri Nets into a data carrying message. This enables the methodology to exhibit currently available data at any stage of the process.

3. Ability to Model Concurrency

The inherent ability of Petri Nets to model concurrency by activating several places as the result of a transition is available in this work and provides a useful construct.

4. Ability to Model Mutual Exclusion

As it was shown in different examples it was possible to model mutual exclusion simply by incorporating a structure of control places into nets which allow only one access to the nets or to of parts of the nets at a time.

B. DISADVANTAGES

The disadvantages of this methodology can be based on the properties of Petri Nets and the intended accuracy of the specifications.

1. Complexity

The given examples, though very simple and small in their nature, exhibit a large complexity in stating the specification. This complexity is largely due to the details involved in the timing of systems and also to the attempt to handle data in the timing. The syntax presented is only one way of representing formal timing specification and it is very tedious for the user to deal with it, therefore a future implementation should provide an user interface which is easy to interact with, preferably in a graphical environment.

2. Difficulty to Model Decisions

Petri Nets by their nature are non-deterministic and so the specification methodology presented suffers from this disadvantage when we are forced to model decisions. The idea introduced of "external agents" helps to deal with this problem.

C. FURTHER RESEARCH TOPICS

This work has been a basic step in showing the possibility of specifying the time behavior of a computer jsystem. As further research topics we suggest the following areas:

- Automation of this methodology to hide the complexity of the specification from the user by providing a graphical interface

- Provision of automated features which allow the user to make inquiries about the specified system, e.g. existence of deadlocks, history of invocations of certain subnets, trace of certain messages, etc.

- Development of tools which are able to analyze and compare the performances of different specifications

- Research in the area of timed Petri Nets where actions can be specified to be performed within a specified time

- Application of this methodology in the area of the newly developed computer systems using transputers and their programming language OCCAM

APPENDIX A: EDITED STATIC SPECIFICATION OF THE ABSTRACT PROCESSOR

This edited specification of an Abstract Processor is based on the work of Yurchak (1984). It uses an improved syntax of Davis and Yurchak (1985) which is considered more meaningful. Also, some minor changes to correct errors have been made. The specification consists of two parts: the replacement statement which provide a shortcut for stating frequently used properties, and the specification of the various resources.

```
replace(X,S)
    "equivrel(X,S);"
with
    "X(i,i) = true();
    X(i,j) = X(j,i);
    implies(and(X(i,j),X(j,k)),X(i,k)) = true();"

replace(X,S)
    "reflexive(x,S);"
with
    "X(i,i) = true();"

replace(X,S)
    "commutative(X,S);"
with
    "X(i,j) = X(j,i);"

replace(X,S)
    "transitive(X,S);"
with
    "implies(and(X(i,j),X(j,k)),X(i,k)) = true();"

replace(X,S)
    "associative(X,S);"
with
    "X(i,X(j,k)) = X(X(i,j),k);"

replace(X,S)
    "irreflexive(X,S);"
```

```
with
    "X(i,i) = false();"

replace(X,S)
    "symmetric(X,S);"
with
    "implies(X(i,j),X(j,i)) = true();"

replace(X,S)
    "antisymmetric(X,S);"
with
    "implies(and(X(i,j),X(j,i)),(i==j)) = true();"

replace(S,T)
    "idopers(S,T);"
with
    "startT: -> S;
    nextT: S -> S;
    prevT: S -> S;
    eqT: S,S -> bool;"

replace(S,T)
    "idaxioms(S,T);"
with
    "prevS(startT()) is undefined;
    prevS(nextS(i)) = i;
    if i != startT()
    then
        nextS(prevS(i)) = i;
    endif;
    equivrel(eqS,S);"

replace(S)
    "typingopers(S);"
with
    "typeS: -> type;
    atomofS: val -> S;
    valofS: S -> val;"

replace(S)
    "typingaxioms(S);"
with
    "whattype(valofS(t)) = typeS();
    atomofS(valofS(t)) = t;
    if whattype(v) = typeS()
    then
        valofS(atomofS(v)) = v;
    else
        atomofS(v) is undefined;
    endif;"

replace(S,T)
```

```
    "relop(S,T);"
with
    "applyrop(ST(),v1,v2) =
        valofbool(TS(atomofS(v1),atomofS(v2)));"

replace(S)
    "isops(S);"
with
    "if whattype(v) = typeS()
    then
        applybop(isS(),v) = valofbool(true());
    else
        applybop(isS(),v) = valofbool(false());
    endif;"

replace(S,T)
    "stateaxioms(S,T);"
with
    "fetchS(a,initam()) is undefined;
    storeS(fetchS(a,q),a,q) = q;
    implies(eqT(a1,a2),fetchS(a1,storeS(v,a2,q)) = v)
        = true();
    implies(not(eqT(a1,a2),fetchS(a1,storeS(v,a2,q)) =
        fetchS(a1,q)) = true();"


Resource boolean is

    Extension of

    Operand Types
        bool;

    Operators
        true: -> bool;
        false: -> bool;
        not: bool -> bool;
        and: bool,bool -> bool;

        Derived Operators
            or: bool,bool -> bool;
            implies: bool,bool -> bool;

        Derived Definition
            or(b1,b2) = not(and(not(b1),not(b2)));
            implies(b1,b2) = not(and(b1,not(b2)));

    Properties
        false = not(true());
        not(not(b)) = b;
        and(true(),b) = b;
        and(false(),b) = false();
```

```
        commutative(and,bool);

end boolean;


Resource natural is

    Extension of
        boolean

    Operand Types
        nat;

    Operators
        zeronat: -> nat;
        prednat: nat -> nat;
        succnat: nat -> nat;
        sumnat: nat,nat -> nat;
        mltnat: nat,nat -> nat;
        divnat: nat,nat -> nat;
        eqnat: nat,nat -> bool;
        gtnat: nat,nat -> bool;

        Derived Operators
            ltnat: nat,nat -> bool;
            genat: nat,nat -> bool;
            lenat: nat,nat -> bool;
            nenat: nat,nat -> bool;

        Derived Definition
            ltnat(n,m) = not(or(gtnat(n,m),eqnat(n,m)));
            genat(n,m) = not(ltnat(n,m);
            lenat(n,m) = not(gtnat(n,m);
            nenat(n,m) = not(eqnat(n,m);

    Properties
        prednat(zeronat()) is undefined;
        prednat(succnat(n)) = n;
        succnat(prednat(n)) = n;
        sumnat(n,zeronat()) = n;
        sumnat(n,succnat(m)) = succnat(sumnat(n,m));
        subnat(n,zeronat()) = n;
        if gtnat(n,m) = true()
        then
            subnat(n,succnat(m)) = prednat(subnat(n,m));
        else
            subnat(n,succnat(m)) is undefined;
        endif;
        mltnat(x,zeronat()) = zeronat();
        mltnat(x,succnat(zeronat())) = x;
        mltnat(x,y) = sumnat(x,mltnat(x,prednat(y)));
        if eqnat(y,zeronat()) = true()
```

```
        then
            divnat(x,y) is undefined;
        else if ltnat(x,y) = true()
        then
            divnat(x,y) = zeronat;
        else
            divnat(x,y) = sumnat(succnat(zeronat(),
                divnat(subnat(x,y),y));
        endif;
        endif;
        eqnat(n,m) = eqnat(succnat(n),succnat(m));
        gtnat(succnat(n),n) = true();
        equivrel(eqnat,nat);
        irreflexive(gtnat,nat);
        irreflexive(ltnat,nat);
        transitive(gtnat,nat);
        transitive(ltnat,nat);
        transitive(genat,nat);
        transitive(lenat,nat);
        antisymmetric(genat,nat);
        antisymmetric(lenat,nat);
        symmetric(nenat,nat);
        commutative(sumnat,nat);
        commutative(mltnat,nat);
        associative(sumnat,nat);
        associative(mltnat,nat);

end natural;


Resource integer is

    Extension of
        boolean,
        natural

    Operand Types
        int;

    Operators
        zeroint: -> int;
        ntoi: nat -> int;
        iton: int -> nat;
        predint: int -> int;
        succint: int -> int;
        sumint: int,int -> int;
        mltint: int,int -> int;
        divint: int,int -> int;
        modint: int,int -> int;
        eqint: int,int -> bool;
        gtint: int,int -> bool;
```

```
    Derived Operators
        ltint: int,int -> bool;
        geint: int,int -> bool;
        leint: int,int -> bool;
        neint: int,int -> bool;

    Derived Definition
        ltint(n,m) = not(or(gtint(n,m),eqint(n,m)));
        geint(n,m) = not(ltint(n,m);
        leint(n,m) = not(gtint(n,m);
        neint(n,m) = not(eqint(n,m);

Properties
    predint(succint(n)) = n;
    succint(predint(n)) = n;
    ntoi(zeronat()) = zeroint();
    ntoi(succnat(n)) =
        sumint(succint(zeroint()),ntoi(n));
    iton(zeroint()) = zeronat();
    if ltint(x,zeroint()) = true()
    then
        iton(x) is undefined;
    else
        iton(succint(x)) = sumnat(
            succnat(zeronat(),iton(x)));
    endif;
    if ltint(x,zeroint()) = true()
    then
        absint(x) = subint(zeroint(),x);
    else
        absint(x) = x;
    endif;
    sumint(n,zeroint()) = n;
    sumint(n,succint(m)) = succint(sumint(n,m));
    subint(n,zeroint()) = n;
    subint(n,succint(m)) = predint(subint(n,m));
        subint(n,succint(m)) is undefined;
    mltint(x,zeroint()) = zeroint();
    mltint(x,succint(zeroint())) = x;
    mltint(x,y) = sumint(x,mltint(x,predint(y)));
    if eqint(y,zeroint()) = true()
    then
        divint(x,y) is undefined;
    else if ltint(absint(x),absint(y)) = true()
    then
        divint(x,y) = zeroint;
    else if or(
            and(gtint(x,zeroint()),
                gtint(y,zeroint())),
            and(ltint(x,zeroint()),
                ltint(y,zeroint()))
            ) = true()
```

```
        then
            divint(x,y) = sumint(succint(zeroint(),
                divint(subint(x,y),y));
        else
            divint(x,y) = sumint(predint(zeroint(),
                divint(sumint(x,y),y));
        endif;
        endif;
        endif;
        if gtint(m,zeroint()) = true()
        then
            if ltint(n,zeroint()) = true()
            then
                modint(n,m) = modint(sumint(n,m),m);
            else
                modint(n,m) = subnat(n,
                    mltnat(m,divint(n,m)));
            endif;
        else
            modint(n,m) is undefined;
        endif;
        eqint(n,m) = eqint(succint(n),succint(m));
        gtint(succint(n),n) = true();
        equivrel(eqint,int);
        irreflexive(gtint,int);
        irreflexive(ltint,int);
        transitive(gtint,int);
        transitive(ltint,int);
        transitive(geint,int);
        transitive(leint,int);
        antisymmetric(geint,int);
        antisymmetric(leint,int);
        symmetric(neint,int);
        commutative(sumint,int);
        commutative(mltint,int);
        associative(sumint,int);
        associative(mltint,int);

end integer;


Resource character is

    Extension of
        boolean

    Operands
        char;

    Operators
        'A','B','C',...,'Z': -> char;
        'a','b','c',...,'z': -> char;
```

```
'!','@','#','$','%','^','&','*','(',')': -> char;
'-','_','+','=','~','`','{','}','[',']': -> char;
''','"',':',';',',','.','<','>','?','/': -> char;
'|','': -> char;
'1','2','3','4','5','6','7','8','9','0': -> char;
NUL: -> char;
SOH: -> char;
STX: -> char;
EXT: -> char;
EOT: -> char;
ENQ: -> char;
ACK: -> char;
BEL: -> char;
BS: -> char;
HT: -> char;
LF: -> char;
VT: -> char;
FF: -> char;
CR: -> char;
SO: -> char;
SI: -> char;
DLE: -> char;
DC1: -> char;
DC2: -> char;
DC3: -> char;
DC4: -> char;
NAK: -> char;
SYN: -> char;
ETB: -> char;
CAN: -> char;
EM: -> char;
SUB: -> char;
ESC: -> char;
FS: -> char;
GS: -> char;
RS: -> char;
US: -> char;
SP: -> char;
DEL: -> char;
eqchar: char,char -> bool;
gtchar: char,char -> bool;

Derived Operators
    ltchar: char,char -> bool;
    gechar: char,char -> bool;
    lechar: char,char -> bool;
    nechar: char,char -> bool;

Derived Definition
    ltchar(n,m) = not(or(gtchar(n,m),eqchar(n,m)));
    gechar(n,m) = not(ltchar(n,m));
    lechar(n,m) = not(gtchar(n,m));
```

```
        nechar(n,m) = not(eqchar(n,m));

Properties
    gtchar(DLE,'~') = true();
    gtchar('~','}') = true();
    gtchar('}','|') = true();
    gtchar('|','{') = true();
    gtchar('{','z') = true();
    gtchar('z',...'a') = true();
    gtchar('a','`') = true();
    gtchar('`','_') = true();
    gtchar('_','^') = true();
    gtchar('^',']') = true();
    gtchar(']','') = true();
    gtchar('','[') = true();
    gtchar('[','Z') = true();
    gtchar('Z',...'A') = true();
    gtchar('A','@') = true();
    gtchar('@','?') = true();
    gtchar('?','>') = true();
    gtchar('>','=') = true();
    gtchar('=','<') = true();
    gtchar('<',';') = true();
    gtchar(';',':') = true();
    gtchar(':','9') = true();
    gtchar('9',...'0') = true();
    gtchar('0','/') = true();
    gtchar('/','.') = true();
    gtchar('.','-') = true();
    gtchar('-',',') = true();
    gtchar(',','+') = true();
    gtchar('+','*') = true();
    gtchar('*',')') = true();
    gtchar(')','(') = true();
    gtchar('(',''') = true();
    gtchar(''','&') = true();
    gtchar('&','%') = true();
    gtchar('%','$') = true();
    gtchar('$','#') = true();
    gtchar('#','"') = true();
    gtchar('"','!') = true();
    gtchar('!',SP) = true();
    gtchar(SP,US) = true();
    gtchar(US,RS) = true();
    gtchar(RS,GS) = true();
    gtchar(GS,FS) = true();
    gtchar(FS,ESC) = true();
    gtchar(ESC,SUB) = true();
    gtchar(SUB,EM) = true();
    gtchar(EM,CAN) = true();
    gtchar(CAN,ETB) = true();
```

```
        gtchar(ETB,SYN) = true();
        gtchar(SYN,NAK) = true();
        gtchar(NAK,DC4) = true();
        gtchar(DC4,DC3) = true();
        gtchar(DC3,DC2) = true();
        gtchar(DC2,DC1) = true();
        gtchar(DC1,DLE) = true();
        gtchar(DLE,SI) = true();
        gtchar(SI,SO) = true();
        gtchar(SO,CR) = true();
        gtchar(CR,FF) = true();
        gtchar(FF,VT) = true();
        gtchar(VT,LF) = true();
        gtchar(LF,HT) = true();
        gtchar(HT,BS) = true();
        gtchar(BS,BEL) = true();
        gtchar(BEL,ACK) = true();
        gtchar(ACK,ENQ) = true();
        gtchar(ENQ,EOT) = true();
        gtchar(EOT,ETX) = true();
        gtchar(ETX,STX) = true();
        gtchar(STX,SOH) = true();
        gtchar(SOH,NUL) = true();
        equivrel(eqchar,char);
        irreflexive(gtchar,char);
        transitive(gtchar,char);
        irreflexive(ltchar,char);
        transitive(ltchar,char);
        transitive(gechar,char);
        transitive(lechar)char);
        antisymmetric(gechar,char);
        antisymmetric(lechar,char);
        symmetric(nechar);

end ;


Resource string(elelment) is

    Parameter element is

        Extension of
            boolean

        Operands
            lm;

        Operators
            eqlm: lm,lm -> bool;
            gtlm: lm,lm -> bool;

            Derived Operators
```

```
        ltlm: lm,lm -> bool;
        gelm: lm,lm -> bool;
        lelm: lm,lm -> bool;
        nelm: lm,lm -> bool;
        Derived Definition
            ltlm(n,m) = not(or(gtlm(n,m),eqlm(n,m)));
            gelm(n,m) = not(ltlm(n,m));
            lelm(n,m) = not(gtlm(n,m));
            nelm(n,m) = not(eqlm(n,m));

    Properties
        equivrel(eqlm,lm);
        irreflexive(gtlm,lm);
        transitive(gtlm,lm);
        irreflexive(ltlm,lm);
        transitive(ltlm,lm);
        transitive(gelm,lm);
        transitive(lelm)lm);
        antisymmetric(gelm,lm);
        antisymmetric(lelm,lm);
        symmetric(nelm);

Extension of
    natural;
    boolean;

Operands
    str.lm;

Operators
    nullstr.lm: -> str.lm;
    makestr.lm: lm -> str.lm;
    lenstr.lm: str.lm -> nat;
    headstr.lm: str.lm -> lm;
    tailstr.lm: str.lm -> str.lm;
    catstr.lm: str.lm,str.lm -> str.lm;
    eqstr.lm: str.lm,str.lm -> bool;
    gtstr.lm: str.lm,str.lm -> bool;

    Derived Operators
        ltstr.lm: str.lm,str.lm -> bool;
        gestr.lm: str.lm,str.lm -> bool;
        lestr.lm: str.lm,str.lm -> bool;
        nestr.lm: str.lm,str.lm -> bool;
    Derived Definition
        ltstr.lm(n,m) =
            not(or(gtstr.lm(n,m),eqstr.lm(n,m)));
        gestr.lm(n,m) = not(ltstr.lm(n,m));
        lestr.lm(n,m) = not(gtstr.lm(n,m));
        nestr.lm(n,m) = not(eqstr.lm(n,m));

Properties
```

```
        lenstr.lm(nullstr.lm()) = zeronat();
        lenstr.lm(makestr.lm(l)) = succnat(zeronat());
        lenstr.lm(catstr.lm(s1,s2)) =
            sumnat(lenstr.lm(s1),lenstr.lm(s2));
        headstr.lm(makestr.lm(l)) = l;
        tailstr.lm(makestr.lm(l)) = nullstr.lm();
        headstr.lm(catstr.lm(makestr.lm(l),s)) = l;
        tailstr.lm(catstr.lm(makestr.lm(l),s)) = s;
        headstr.lm(nullstr.lm()) is undefined;
        tailstr.lm(nullstr.lm()) = nullstr.lm();
        catstr.lm(catstr.lm(s1,s2),s3) =
            catstr.lm(s1,catstr.lm(s2,s3));
        catstr.lm(nullstr.lm(),s) = catstr.lm(
            s,nullstr.lm()) = s;
        implies(eqlm(l1,l2),eqstr.lm(makestr.lm(l1),
            makestr.lm(l2))) = true();
        implies(gtlm(l1,l2),gtstr.lm(makestr.lm(l1),
            makestr.lm(l2))) = true();
        gtnat(lenstr.lm(makestr.lm(l),
            lenstr.lm(nullstr.lm()) = true();
        implies(gtnat(lenstr.lm(s1),lenstr.lm(s2)),
            gtstr.lm(s1,s2) = true();
        if lenstr.lm(s1) != zeronat()
        then
            gtnat(lenstr.lm(catstr.lm(s1,s2),
            lenstr.lm(s2) = true();
        else
            eqnat(lenstr.lm(catstr.lm(s1,s2),
            lenstr.lm(s2) = true();
        endif;
        equivrel(eqstr.lm,str.lm);
        irreflexive(gtstr.lm,str.lm);
        transitive(gtstr.lm,str.lm);
        irreflexive(ltstr.lm,str.lm);
        transitive(ltstr.lm,str.lm);
        transitive(gestr.lm,str.lm);
        transitive(lestr.lm)str.lm);
        antisymmetric(gestr.lm,str.lm);
        antisymmetric(lestr.lm,str.lm);
        symmetric(nestr.lm,str.lm);

end string(element);


Resource string(character) is

    where
        lm = char;

        nullstr.char = nullstr.lm;
        makestr.char = makestr.lm;
        len.char = lenstr.lm;
```

```
        head.char = headstr.lm;
        tail.char = tailstr.lm;
        cat.char = catstr.lm;
        eq.char = eqstr.lm;
        gt.char = gtstr.lm;
        lt.char = ltstr.lm;
        ge.char = gestr.lm;
        le.char = lestr.lm;
        ne.char = nestr.lm;

end string(character);


Resource identifiers is

    Extension of
        boolean

    Operands
        memid;
        regid;
        stkid;
        fid;

    Operators
        idopers(memid,memsem);
        idopers(redid,regseg);
        idopers(stkid,stkseg);
        idopers(fid.fseg);

    Properties
        idaxiom(memid,memseg);
        idaxiom(regid,regseg);
        idaxiom(stkid,stkseg);
        idaxiom(fid,fseg);

end ;


Resource memaddress is

    Extension of
        identifiers,
        boolean

    Operands
        memaddr;

    Operators
        startmemaddr: memid -> memaddr;
        nextmemaddr: memaddr -> memaddr;
        prevmemaddr: memaddr -> memaddr;
```

```
        getmemid: memaddr -> memid;
        offset: int,memaddr -> memaddr;
        eqmemaddr: memaddr,memaddr -> bool;

    Properties
        prevmemaddr(startmemaddr(i)) is undefined;
        prevmemaddr(nextmemaddr(m)) = m;
        nextmemaddr(prevmemaddr(m)) = m;
        offset(succint(n),m) = nextmemaddr(offset(n,m));
        if offset(n,m) = startmemaddr()
        then
            offset(predint(n),m) is undefined;
        else
            offset(predint(n),m) = prevmemaddr(offset(n,m));
        endif;
        eqmemid(i,getmemid(offset(n,startmemaddr(i)))) =
            true();
        eqmemaddr(startmemaddr(i1),startmemaddr(i2)) =
            eqmemid(i1,i2);
        eqmemaddr(startmemaddr(i),nextmemaddr(a)) = false();
        eqmemaddr(nextmemaddr(a1),nextmemaddr(a2)) =
            eqmemaddr(a1,a2);
        offset(zeroint(),m) = m;
        eqivrel(eqmemaddr,memaddr);

end memaddress ;


Resource regaddress is

    Extension of
        identifiers,
        boolean

    Operands
        regaddr;

    Operators
        startregaddr: regid -> regaddr;
        nextregaddr: regaddr -> regaddr;
        prevregaddr: regaddr -> regaddr;
        getregid: regaddr -> regid;
        eqregaddr: regaddr,regaddr -> bool;

    Properties
        prevregaddr(startregaddr(i)) is undefined;
        prevregaddr(nextregaddr(m)) = m;
        nextregaddr(prevregaddr(m)) = m;

        eqregaddr(startregaddr(i1),startregaddr(i2)) =
            eqregid(i1,i2);
        eqregaddr(startregaddr(i),nextregaddr(a)) = false();
```

```
        eqregaddr(nextregaddr(a1),nextregaddr(a2)) =
            eqregaddr(a1,a2);

        eqivrel(eqregaddr,regaddr);

end regaddress;


Resource stkaddress is

    Extension of
        identifiers,
        boolean

    Operands
        stkaddr;

    Operators
        getstkid: stkaddr -> stkid;
        eqstkaddr: stkaddr,atkaddr -> bool;

    Properties
        eqstkaddr(nextstkaddr(a1),nextstkaddr(a2)) =
            eqstkaddr(a1,a2);
        equivrel(eqstkaddr,stkaddr);

end stkaddress ;


Resource files is

    Extension of
        identifiers,
        boolean

    Operands
        file;

    Operators
        getfile: fid -> file;
        eqfile: file,file -> bool;

    Properties
        eqfile(getfile(i1),getfile(i2)) = eqfile(i1,i2);
        equivrel(eqfile,file);

end files;

Resource operatorclasses is

    Extension of
```

```
    Operands
        mop;
        dop;
        top;
        qop;
        sop;
        oop;
        rop;
        bop;

    Operators

    Properties

end operatorclasses;

Resource instructiontype is

    Extension of

    Operands
        instr;

    Operators

    Properties

end intructiontype;

Resource typing is

    Extension of
        boolean;
        natural;
        integer;
        character;
        string(character);
        identifiers;
        memaddress;
        regaddress;
        stkaddr;
        files;
        operatorclasses;
        intructiontype;

    Operands
        type;
        val;

    Operators
        typingopers(bool);
        typingopers(nat);
```

```
        typingopers(int);
        typingopers(char);
        typingopers(str.char));
        typingopers(memid);
        typingopers(regid);
        typingopers(stkid);
        typingopers(fid);
        typingopers(memaddr);
        typingopers(regaddr);
        typingopers(stkaddr);
        typingopers(file);
        typingopers(mop);
        typingopers(dop);
        typingopers(top);
        typingopers(qop);
        typingopers(sop);
        typingopers(oop);
        typingopers(rop);
        typingopers(bop);
        typingopers(instr);

        whattype: val -> type;
        eqtype: type,type -> bool;

    Properties
        typingaxiom(bool);
        typingaxiom(nat);
        typingaxiom(int);
        typingaxiom(char);
        typingaxiom(str.char);
        typingaxiom(memid);
        typingaxiom(regid);
        typingaxiom(stkid);
        typingaxiom(fid);
        typingaxiom(memaddr);
        typingaxiom(regaddr);
        typingaxiom(stkaddr);
        typingaxiom(file);
        typingaxiom(mop);
        typingaxiom(dop);
        typingaxiom(top);
        typingaxiom(qop);
        typingaxiom(sop);
        typingaxiom(oop);
        typingaxiom(rop);
        typingaxiom(bop);
        typingaxiom(instr);
        equivrel(instr);

end typing;
```

```
Resource operators is

    Extension of
        operatorclasses,
        typing

    Operands

    Operators
        boolnot: -> mop;
        booland: -> dop;
        boolor: -> dop;
        natpred: -> mop;
        natsucc: -> mop;
        natsum: -> dop;
        natsub: -> dop;
        nateq: -> rop;
        natgt: -> rop;
        natlt: -> rop;
        intpred: -> mop;
        intsucc: -> mop;
        intabs: -> mop;
        intntoi: -> mop;
        intiton: -> mop;
        intsum: -> dop;
        intsub: -> dop;
        intmlt: -> dop;
        intdiv: -> dop;
        intmod: -> dop;
        inteq: -> rop;
        intgt: -> rop;
        intlt: -> rop;
        chareq: -> rop;
        chargt: -> rop;
        charmakestr: -> mop;
        charstrlen: -> mop;
        charheadstr: -> mop;
        chartailstr: -> mop;
        charcatstr: -> dop;
        str.chareq: -> rop;
        str.chargt: -> rop;
        isbool: -> bop;
        isnat: -> bop;
        isint: -> bop;
        ischar: -> bop;
        isstr.char: -> bop;
        ismemid: -> bop;
        isregid: -> bop;
        isstkid: -> bop;
        isfid: -> bop;
        ismemaddr: -> bop;
        isregaddr: -> bop;
```

```
    isstkaddr: -> bop;
    isfile: -> bop;
    ismop: -> bop;
    isdop: -> bop;
    istop: -> bop;
    isqop: -> bop;
    issop: -> bop;
    isoop: -> bop;
    isrop: -> bop;
    isbop: -> bop;
    isinstr: -> bop;

    applymop: mop,val -> val;
    applydop: dop,val,val -> val;
    applytop: top,val,val,val -> val;
    applyqop: qop,val,val,val,val -> val;
    applysop: sop,val,val,val,val,val -> val;
    applyoop: oop,val,val,val,val,val,val -> val;
    applyrop: rop,val,val -> val;
    applybop: bop,val -> val;


Properties
    applymop(boolnot(),v) = valofbool(not(atomofbool(v));
    applydop(booland(),v1,v2) = valofbool(
        and(atomofbool(v1),atomofbool(v2));
    applydop(boolor(),v1,v2) = valofbool(
        or(atomofbool(v1),atomofbool(v2));
    applymop(natpred(),v) = valofnat(
        prednat(atomofnat(v));
    applymop(natsucc(),v) = valofnat(
        succnat(atomofnat(v));
    applydop(natsum(),v1,v2) = valofnat(
        sumnat(atomofnat(v1),atomofnat(v2));
    applydop(natsub(),v1,v2) = valofnat(
        subnat(atomofnat(v1),atomofnat(v2));
    applymop(intpred(),v) = valofint(
        predint(atomofint(v));
    applymop(intsucc(),v) = valofint(
        succint(atomofint(v));
    applymop(intabs(),v) = valofint(
        absint(atomofint(v));
    applymop(intntoi(),v) = valofint(
        ntoiint(atomofint(v));
    applymop(intiton(),v) = valofint(
        itonint(atomofint(v));
    applydop(intsum(),v1,v2) = valofint(
        sumint(atomofint(v1),atomofint(v2));
    applydop(intsub(),v1,v2) = valofint(
        subint(atomofint(v1),atomofint(v2));
    applydop(intmlt(),v1,v2) = valofint(
        mltint(atomofint(v1),atomofint(v2));
```

```
        applydop(intdiv(),v1,v2) = valofint(
            divint(atomofint(v1),atomofint(v2));
        applydop(intmod(),v1,v2) = valofint(
            modint(atomofint(v1),atomofint(v2));
        applymop(charstrlen(),v) = valofnat(
            lenstr.char(atomofstr.char(v)));
        applymop(charmakestr(),v) = valofstr.char(
            makestr.char(atomofchar(v)));
        applymop(charheadstr(),v) = valofchar(
            headstr.char(atomofstr.char(v)));
        applymop(chartailstr(),v) = valofstr.char(
            tailstr.char(atomofstr.char(v)));
        applydop(charcatstr(),v1,v2) = valofstr.char(
            catstr.char(atomofstr.char(v1),
            atomofstr.char(v2)));
        relop(nat,eq);
        relop(nat,gt);
        relop(nat,lt);
        relop(int,eq);
        relop(int,gt);
        relop(int,lt);
        relop(char,eq);
        relop(char,gt);
        relop(str.char,eq);
        relop(str.char,gt);
        isops(bool);
        isops(nat);
        isops(int);
        isops(char);
        isops(str.char);
        isops(memid);
        isops(regid);
        isops(stkid);
        isops(fid);
        isops(memaddr);
        isops(regaddr);
        isops(stkaddr);
        isops(file);
        isops(mop);
        isops(dop);
        isops(top);
        isops(qop);
        isops(sop);
        isops(oop);
        isops(rop);
        isops(bop);
        isops(instr);


end operators;
```

```
Resource intructions is

    Extension of
        natural,
        integer,
        memaddress,
        regaddress,
        stkaddress,
        operatorclasses,
        intructiontype,
        typing

    Operands

    Operators
        org: -> instr;
        extern: -> instr;
        globl: -> instr;
        mbegin: -> instr;
        mend: -> instr;
        offst: int,regaddr -> instr;
        link: regaddr,nat -> instr;
        unlink: regaddr -> instr;
        monads: mop,regaddr -> instr;
        monad: mop,regaddr,regaddr -> instr;
        monadi: mop,val,regaddr -> instr;
        dyads: dop,regaddr,regaddr -> instr;
        dyadsi: dop,val,regaddr -> instr;
        dyad: dop,regaddr,regaddr,regaddr -> instr;
        dyadi: dop,val,regaddr,regaddr -> instr;
        triads: top,regaddr,regaddr,regaddr -> instr;
        triadsi: top,val,regaddr,regaddr -> instr;
        triad: top,regaddr,regaddr,regaddr,regaddr -> instr;
        triadi: top,val,regaddr,regaddr,regaddr -> instr;
        quads: qop,regaddr,regaddr,regaddr,regaddr -> instr;
        quad: qop,regaddr,regaddr,regaddr,
            regaddr,regaddr -> instr;
        sexads: sop,regaddr,regaddr,regaddr,
            regaddr,regaddr,regaddr -> instr;
        sexad: sop,regaddr,regaddr,regaddr,regaddr,
            regaddr,regaddr,regaddr -> instr;
        octads: oop,regaddr,regaddr,regaddr,regaddr,
            regaddr,regaddr,regaddr,regaddr -> instr;
        octad: oop,regaddr,regaddr,regaddr,regaddr,
            regaddr,regaddr,regaddr,regaddr,regaddr -> instr;
        movi_m: val,memaddr -> instr;
        movi_pcr: val,int -> instr;
        movi_r: val,regaddr -> instr;
        movi_ri: val,regaddr -> instr;
        movi_rid: val,regaddr,int -> instr;
        movi_ridn: val,regaddr,nat,int -> instr;
        mov_m_m: memaddr,memaddr -> instr;
```

```
mov_m_r: memaddr,regaddr -> instr;
mov_m_ri: memaddr,regaddr -> instr;
mov_m_rid: memaddr,regaddr,int -> instr;
mov_m_ridn: memaddr,regaddr,nat,int -> instr;
mov_pcr_pcr: int,int -> instr;
mov_pcr_r: int,regaddr -> instr;
mov_pcr_ri: int,regaddr -> instr;
mov_pcr_rid: int,regaddr,int -> instr;
mov_pcr_ridn: int,regaddr,nat,int -> instr;
mov_r_m: regaddr,memaddr -> instr;
mov_r_pcr: regaddr,int -> instr;
mov_r_r: regaddr,regaddr -> instr;
mov_r_ri: regaddr,regaddr -> instr;
mov_r_rid: regaddr,regaddr,int -> instr;
mov_r_ridn: regaddr,regaddr,nat,int -> instr;
mov_ri_m: regaddr,memaddr -> instr;
mov_ri_pcr: regaddr,int -> instr;
mov_ri_r: regaddr,regaddr -> instr;
mov_ri_ri: regaddr,regaddr -> instr;
mov_ri_rid: regaddr,regaddr,int -> instr;
mov_ri_ridn: regaddr,regaddr,nat,int -> instr;
mov_rid_m: regaddr,int,memaddr -> instr;
mov_rid_pcr: regaddr,int,int -> instr;
mov_rid_r: regaddr,int,regaddr -> instr;
mov_rid_ri: regaddr,int,regaddr -> instr;
mov_rid_rid: regaddr,int,regaddr,int -> instr;
mov_rid_ridn: regaddr,int,regaddr,nat,int -> instr;
mov_ridn_m: regaddr,nat,int,memaddr -> instr;
mov_ridn_pcr: regaddr,nat,int,int -> instr;
mov_ridn_r: regaddr,nat,int,regaddr -> instr;
mov_ridn_ri: regaddr,nat,int,regaddr -> instr;
mov_ridn_rid: regaddr,nat,int,regaddr,int -> instr;
mov_ridn_ridn: regaddr,nat,int,regaddr,nat,
    int -> instr;
push_i: val,stkaddr -> instr;
push_m: memaddr,stkaddr -> instr;
push_pcr:   int,stkaddr -> instr;
push_r: regaddr,stkaddr -> instr;
push_ri: regaddr,stkaddr -> instr;
push_rid: regaddr,int,stkaddr -> instr;
push_ridn: regaddr,nat,int,stkaddr -> instr;
pop_x: stkaddr -> instr;
pop_m: stkaddr,memaddr -> instr;
pop_pcr: stkaddr,int -> instr;
pop_r: stkaddr,regaddr -> instr;
pop_ri: stkaddr,regaddr -> instr;
pop_rid: stkaddr,regaddr,int -> instr;
pop_ridn: stkaddr,regaddr,nat,int -> instr;
nop: -> instr;
stop: -> instr;
jmp: memaddr -> instr;
jmp_i: memaddr -> instr;
```

```
        jmp_r: regaddr -> instr;
        bra: int -> instr;
        bra_r: regaddr -> instr;
        if: relop,regaddr,regaddr,memaddr -> instr;
        ifi: relop,regaddr,val,memaddr -> instr;
        ifte: relop,regaddr,regaddr,memaddr,memaddr -> instr;
        iftei: relop,regaddr,val,memaddr,memaddr -> instr;
        if_pcr: relop,regaddr,regaddr,int -> instr;
        ifi_pcr: relop,regaddr,val,int -> instr;
        ifte_pcr: relop,regaddr,regaddr,int,int -> instr;
        iftei_pcr: relop,regaddr,val,int,int -> instr;
        test: bop,regaddr,memaddr -> instr;
        testm: bop,memaddr,memaddr -> instr;
        teste: bop,regaddr,memaddr,memaddr -> instr;
        testme: bop,memaddr,memaddr,memaddr -> instr;
        test_pcr: bop,regaddr,int -> instr;
        testm_pcr: bop,memaddr,int -> instr;
        teste_pcr: bop,regaddr,int,int -> instr;
        testme_pcr: bop,memaddr,int,int -> instr;
        jsr: memaddr,stkaddr -> instr;
        jsr_i: memaddr,stkaddr -> instr;
        jsr_r: regaddr,stkaddr -> instr;
        bsr: int,stkaddr -> instr;
        bsr-r: regaddr,stkaddr -> instr;
        rts: stkaddr -> instr;
        open: stkaddr -> instr;
        close: stkaddr -> instr;
        read: stkaddr -> instr;
        write: stkaddr -> instr;

    Properties

end intructions;


Resource amstate is

    Extension of
        boolean,
        natural,
        integer,
        str(character),
        memaddress,
        regaddress,
        files,
        identifiers,
        typing

    Operands
        state;

    Operators
```

```
    fetchm: memaddr,state -> val;
    fetchr: regaddr,state -> val;
    storem: val,memaddr,state -> state;
    storer: val,regaddr,state -> state;
    initam: -> state;
    initstk: stkaddr,state -> state;
    topstk: stkaddr,state -> val;
    pushstk: val,stkaddr,state -> state;
    popstk: stkaddr,state -> state;
    lalloc: nat,state -. memid;
    lfree: memid,state -> state;
    indir: nat,memaddr -> memaddr;
    infile: file,state -> val;
    outfile: val,file,state -> state;
    openfile: str.char,file,int,int,state -> state;
    closefile: file,state -> state;
    rmode: -> int;
    wmode: -> int:
    rwmode: -> int:
    openerr: -> int;
    openok: -> int;
    valdata: -> int;
    chardata: -> int;

    active: memid,state -> bool;

Properties
    topstk(s,initstk(s)) is undefined;
    popstk(s,initstk(s)) is undefined;
    popstk(s,initam()) is undefined;
    stateaxiom(m,memaddr);
    stateaxiom(r,regaddr);
    topstk(s(pushstk(v,s,q)) = v;
    popstk(s(pushstk(v,s,q)) = q;
    active(m,initam()) = false();
    active(lalloc(n,q)q) = true();
    actice(m,lfree(m,q)) = false();
    active(m,storer(v,r,q)) = active(m,q);
    active(m,storem(v,a,q)) = active(m,q);
    active(m,initstk(s,q)) = active(m,q);
    active(m,pushstk(v,s,q) = active(m,q);
    active(m,popstk(s,q) = active(m,q);
    active(m,outfile(v,f,q) = active(m,q);
    active(m,openfile(s,f,x,y,q) = active(m,q);
    active(m,closefile(f,q) = active(m,q);
    if active(m,q) = false()
    then
        fetchm(offset(n,m),q) is undefined;
    endif;
    if active(m,q) = false();
    then
        storem(v,offset(n,m),q) is undefined;
```

```
        endif;
        if ltint(n,ntoi(n2)) = true()
        then
            offset(n,offset(n1,startmemaddr(
                lalloc(n2,q)))) =
                offset(sumint(n,n1),
                startmemaddr(lalloc(n2,q)));
        else
            offset(n,offset(n1,startmemaddr(
                lalloc(n2,q)))) is undefined;
        endif;
        indir(zeronat(),m) = m;
        if whattype(fetchm(indir(n,m),q) = typememaddr()
        then
            indir(succnat(n),m) =
                atomofmemaddr(fetchm(indir(n,m),q));
        else
            indir(succnat(n),m) is undefined;
        endif;
        openfile(s,f,n,openfile(s,f,m,x,q)) is undefined;
        closefile(f(openfile(s,f,n,x,q)) = q;
        infile(f,initam()) is undefined;
        infile(f,close(f,q)) is undefined;
        infile(f,openfile(s,f,wmode(),x,q)) is undefined;
        outfile(v,f,initam()) is undefined;
        outfile(v,f,close(f,q)) is undefined;
        outfile(v,f,openfile(s,f,rmode(),x,q)) is undefined;
        outifle(v,f,openfile(s,f,m,chardata(),q))
            is undefined;


end amstate;


Resource am is

    Extension of
        memaddress,
        intructiontype,
        typing,
        amstate

    Operands
        prog: memaddr,state -> state;

        cond: val,memaddr,memaddr - memaddr;
        exq: instr,memaddr,state -> state;

    Operators
        prog(m,q) = exq(atomofinstr(fetchm(m,q)),m,q);
        cond(valofbool(true(),m1,m2)) = m1;
        cond(valofboll(false(),m1,m2)) = m2;
```

```
Properties
    exq(offst(i,r),m,q) =
        prog(nextmemaddr(m),
            storer(
                valofmemaddr(offset(
                    i.atomofmemaddr(fetchr(r,q)))),
            r,q));
    exq(link(r,n)m,q) =
        prog(nextmemaddr(m),
            storer(
                valofmemaddr(startmemaddr(lalloc(n,q))),
                r,storem(fetchr(r,q),
                    startmemaddr(lalloc(n,q),q))));
    exq(unlink(r),m,q) =
        prog(nextmemaddr(m),
            lfree(
                getmemid(atomofmemaddr(fetch(r,q))),
                storer(
                    fetchm(atomofmemaddr(fetchr(r,q),q),
                    r,q)));
    exq(monads(o,r1),m,q) =
        prog(nextmemaddr(m),
            storer(applymop(o,fetchr(r1,q)),
                r1,q));
    exq(monad(o,r1,r2),m,q) =
        prog(nextmemaddr(m),
            storer(applymop(o,fetchr(r1,q)),
                r2,q));
    exq(monadi(o,v,r1),m,q) =
        prog(nextmemaddr(m),
            storer(applymop(o,v)),r1,q));
    exq(dyads(o,r1,r2),m,q) =
        prog(nextmemaddr(m),
            storer(applydop(
                o,fetchr(r1,q),fetchr(r2,q)),r2,q));
    exq(dyadsi(o,v,r1)m,q) =
        prog(nextmemaddr(m),
            store(applydop(v,fetchr(r1,q)),r1,q));
    exq(dyad(o,r1,r2,r3),m,q) =
        prog(nextmemaddr(m),
            storer(applydop(o,
                fetchr(r1,q),fetchr(r2,q)),r3,q));
    exq(dyadi(o,v,r1,r2),m,q) =
        prog(nextmemaddr(m),
            storer(applydop(o,
                v,fetchr(r1,q)),r2,q));
    exq(triads(o,r1,r2,r3),m,q) =
        prog(nextmemaddr(m),
            storer(applytop(o,fetchr(r1,q),
                fetchr(r2,q),fetchr(r3,q)),r3,q));
    exq(triadsi(o,v,r1,r2),m,q) =
```

```
    prog(nextmemaddr(m),
        storer(applytop(o,v,
            fetchr(r1,q),fetchr(r2,q)),r2,q));
exq(triad(o,r1,r2,r3,r4),m,q) =
    prog(nextmemaddr(m),
        storer(applytop(o,fetchr(r1,q),
            fetchr(r2,q),fetchr(r3,q)),r4,q));
exq(triadi(o,v,r1,r2,r3),m,q) =
    prog(nextmemaddr(m),
        storer(applytop(o,v,
            fetchr(r1,q),fetchr(r2,q)),r3,q));
exq(quads(o,r1,r2,r3,r4),m,q) =
    prog(nextmemaddr(m),
        storer(applyqop(o,fetchr(r1,q),
            fetchr(r2,q),fetchr(r3,q),
            fetchr(r4,q),r4,q));
exq(quad(o,r1,r2,r3,r4,r5),m,q) =
    prog(nextmemaddr(m),
        storer(applyqop(o,fetchr(r1,q),
            fetchr(r2,q),fetchr(r3,q),
            fetchr(r4,q),r5,q));
exq(sexads(o,r1,r2,r3,r4,r5,r6),m,q) =
    prog(nextmemaddr(m),
        storer(applyqop(o,fetchr(r1,q),
            fetchr(r2,q),fetchr(r3,q),
            fetchr(r4,q),fetchr(r5,q),
            fetch(r6,q),r6,q));
exq(sexad(o,r1,r2,r3,r4,r5,r6,r7),m,q) =
    prog(nextmemaddr(m),
        storer(applyqop(o,fetchr(r1,q),
            fetchr(r2,q),fetchr(r3,q),
            fetchr(r4,q),fetchr(r5,q),
            fetch(r6,q),r7,q));
exq(octads(o,r1,r2,r3,r4,r5,r6,r7,r8),m,q) =
    prog(nextmemaddr(m),
        storer(applyqop(o,fetchr(r1,q),
            fetchr(r2,q),fetchr(r3,q),
            fetchr(r4,q),fetchr(r5,q),
            fetch(r6,q),fetchr(r7,q),
            fetchr(r8,q),r8,q));
exq(octad(o,r1,r2,r3,r4,r5,r6,r7,r8,r9),m,q) =
    prog(nextmemaddr(m),
        storer(applyqop(o,fetchr(r1,q),
            fetchr(r2,q),fetchr(r3,q),
            fetchr(r4,q),fetchr(r5,q),
            fetch(r6,q),fetchr(r7,q),
            fetchr(r8,q),r9,q));
exq(movi_m(v,m1),m,q) =
    prog(nextmemaddr(m),
        storem(v,m1,q));
exq(movi_pcr(v,i),m,q) =
    prog(nextmemaddr(m),
```

```
        storem(v,offset(i,m),q));
exq(movi_r(v,r),m,q) =
    prog(nextmemaddr(m),
        storem(v,r,q));
exq(movi_ri(v,r),m,q) =
    prog(nextmemaddr(m),
        storem(v,atomofmemaddr(fetchr(r,q),q));
exq(movi_rid(v,r,n),m,q) =
    prog(nextmemaddr(m),
        storem(v,offset(
        n,atomofmemaddr(fetchr(r,q),q));
exq(movi_ridn(v,r,i1,i2),m,q) =
    prog(nextmemaddr(m),
        storem(v,offset(i2,indir(
        i1,atomofmemaddr(fetchr(r,q),q));

exq(mov_m_m(m1,m2),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(m1,q),m2,q));
exq(mov_m_r(m1,r),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(m1,q),r,q));
exq(mov_m_ri(m1,r),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(m1,q),
        atomofmemaddr(fetchr(r,q),q));
exq(mov_m_rid(m1,r,n),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(m1,q),offset(
        n,atomofmemaddr(fetchr(r,q),q));
exq(mov_m_ridn(m1,r,i1,i2),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(m1,q),offset(i2,indir(
        i1,atomofmemaddr(fetchr(r,q),q));

exq(mov_pcr_pcr(i1,i2),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(offset(i1,m),q),
        offset(i2,m),q));
exq(mov_pcr_r(i1,r),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(offset(i1,m),q),r,q));
exq(mov_pcr_ri(i1,r),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(offset(i1,m),q),
        atomofmemaddr(fetchr(r,q),q));
exq(mov_pcr_rid(i1,r,i2),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(offset(i1,m),q),offset(
        i2,atomofmemaddr(fetchr(r,q),q));
exq(mov_pcr_ridn(i1,r,n,i2),m,q) =
    prog(nextmemaddr(m),
```

```
        storem(fetchm(offset(i1,m),q),offset(i2,
        indir(n,atomofmemaddr(fetchr(r,q),q));

exq(mov_r_m(r1,m1),m,q) =
    prog(nextmemaddr(m),
        storem(fetchr(r1,q),m1,q));
exq(mov_r_pcr(r1,i),m,q) =
    prog(nextmemaddr(m),
        storem(fetchr(r1,q),offset(i,m),q));
exq(mov_r_r(r1,r2),m,q) =
    prog(nextmemaddr(m),
        storem(fetchr(r1,q),r2,q));
exq(mov_r_ri(r1,r2),m,q) =
    prog(nextmemaddr(m),
        storem(fetchr(r1,q),
        atomofmemaddr(fetchr(r2,q),q));
exq(mov_r_rid(r1,r2,n),m,q) =
    prog(nextmemaddr(m),
        storem(fetchr(r1,q),offset(
        n,atomofmemaddr(fetchr(r2,q),q));
exq(mov_r_ridn(r1,r2,i1,i2),m,q) =
    prog(nextmemaddr(m),
        storem(fetchr(r1,q),offset(i2,indir(
        i1,atomofmemaddr(fetchr(r2,q),q));

exq(mov_ri_m(r1,m1),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(atomofmemaddr(fetchr(r1,q),q),
        m1,q));
exq(mov_ri_pcr(r1,i),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(atomofmemaddr(fetchr(r1,q),q),
        offset(i,m),q));
exq(mov_ri_r(r1,r2),m,q) =
    prog(nextmemaddr(m),
        storer(fetchm(atomofmemaddr(fetchr(r1,q),q),
        r2,q));
exq(mov_ri_ri(r1,r2),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(atomofmemaddr(fetchr(r1,q),q),
        atomofmemaddr(fetchr(r2,q)),q));
exq(mov_ri_rid(r1,r2,n),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(atomofmemaddr(fetchr(r1,q),q),
        offset(n,atomofmemaddr(fetchr(r2,q)),q));
exq(mov_ri_ridn(r1,r2,i1,i2),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(atomofmemaddr(fetchr(r1,q),q),
        offset(i2,indir(
        i1,atomofmemaddr(fetchr(r2,q))),q));

exq(mov_rid_m(r1,i1,m1),m,q) =
```

```
    prog(nextmemaddr(m),
        storem(fetchm(offset(i1,
        atomofmemaddr(fetchr(r1,q))),q),m1,q));
exq(mov_rid_pcr(r1,i1,i2),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(offset(i1,
        atomofmemaddr(fetchr(r1,q))),q),
        offset(i2,m),q));
exq(mov_rid_r(r1,n,r2),m,q) =
    prog(nextmemaddr(m),
        storer(fetchm(offset(n,
        atomofmemaddr(fetchr(r1,q))),q),r2,q));
exq(mov_rid_ri(r1,i,r2),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(offset(i,
        atomofmemaddr(fetchr(r1,q))),q),
        atomofmemaddr(fetchr(r2,q)),q));
exq(mov_rid_rid(r1,i1,r2,i1),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(offset(i1,
        atomofmemaddr(fetchr(r1,q))),q),
        offset(i1,atomofmemaddr(fetchr(r2,q)),q));
exq(mov_rid_ridn(r1,r2,i1,i2,i3),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(offset(i1,
        atomofmemaddr(fetchr(r1,q))),q),
        offset(i3,indir(
        i2,atomofmemaddr(fetchr(r2,q)))),q));

exq(mov_ridn_m(r1,n,i1,m1),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(offset(i1,
        indir(n,atomofmemaddr(
        fetchr(r1,q)))),q),m1,q));
exq(mov_ridn_pcr(r1,n,i1,i2),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(offset(i1,
        indir(n,atomofmemaddr(fetchr(r1,q)))),q),
        offset(i2,m),q));
exq(mov_ridn_r(r1,i1,i2,r2),m,q) =
    prog(nextmemaddr(m),
        storer(fetchm(offset(i2,
        indir(i1,atomofmemaddr(
        fetchr(r1,q)))),q),r2,q));
exq(mov_ridn_ri(r1,i1,i2,r2),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(offset(i2,
        indir(i1,atomofmemaddr(fetchr(r1,q)))),q),
        atomofmemaddr(fetchr(r2,q)),q));
exq(mov_ridn_rid(r1,i1,i2,r2,i3),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(offset(i2,
```

```
        indir(i1,atomofmemaddr(fetchr(r1,q)))),q),
        offset(i3,atomofmemaddr(fetchr(r2,q)),q));
exq(mov_ridn_ridn(r1,r2,i1,i2,i3,i4),m,q) =
    prog(nextmemaddr(m),
        storem(fetchm(offset(i2,
        indir(i1,atomofmemaddr(fetchr(r1,q)))),q),
        offset(i4,indir(
        i3,atomofmemaddr(fetchr(r2,q)))),q));
exq(push_i(v,s),m,q) =
    prog(nextmemaddr(m),
        pushstk(v,s,q));
exq(push_m(m1,s),m,q) =
    prog(nextmemaddr(m),
        pushstk(fetchm(m1,q),s,q));
exq(push_pcr(i,s),m,q) =
    prog(nextmemaddr(m),
        pushstk(fetchm(offset(i,m),q),s,q));
exq(push_r(r,s),m,q) =
    prog(nextmemaddr(m),
        pushstk(fetchr(r,q),s,q));
exq(push_ri(r,s),m,q) =
    prog(nextmemaddr(m),
        pushstk(atomofmemaddr(fetchr(r,q)),s,q));
exq(push_rid(r,n,s),m,q) =
    prog(nextmemaddr(m),
        pushstk(fetchm(offset(n,
        atomofmemaddr(fetchr(r,q))),q),s,q));
exq(push_ridn(r,i1,i2,s),m,q) =
    prog(nextmemaddr(m),
        pushstk(fetchm(offset(i2,indir(i1,
        atomofmemaddr(fetchr(r,q)))),q),s,q));
exq(pop_x(s),m,q) =
    prog(nextmemaddr(m),
        popstk(s,q));
exq(pop_m(s,m1),m,q) =
    prog(nextmemaddr(m),
        popstk(s,storem(topstk(s,q),m1,q)));
exq(pop_pcr(s,i),m,q) =
    prog(nextmemaddr(m),
        popstk(s,storem(topstk(s,q),
        offset(i,m),q)));
exq(pop_r(s,r),m,q) =
    prog(nextmemaddr(m),
        popstk(s,storer(topstk(s,q),r,q)))),m,q);
exq(pop_ri(s,r),m,q) =
    prog(nextmemaddr(m),
        popstk(s,storem(topstk(s,q),
        atomofmemaddr(fetchr(r,q)),q)));
exq(pop_rid(s,r,n),m,q) =
    prog(nextmemaddr(m),
        popstk(s,storem(topstk(s,q),offset(n,
        atomofmemaddr(fetchr(r,q))),q)));
```

```
exq(pop_ridn(s,r,i1,i2),m,q) =
    prog(nextmemaddr(m),
        popstk(s,storem(topstk(s,q),offset(i2,
        indir(i1,atomofmemaddr(fetchr(r,q)))),q)));
exq(nop,m,q) =
    prog(nextmemaddr(m),q);
exq(stop,m,q) =
    prog(m,q) = q;
exq(jmp(m1),m,q) =
    prog(m1,q);
exq(jmp_i(m1),m,q) =
    prog(atomofmemaddr(fetchm(m1,q)),q);
exq(jmp_r(r),m,q) =
    prog(atomofmemaddr(fetchr(r,q)),q);
exq(bra(n),m,q) =
    prog(offset(n,nextmemaddr(m)),q);
exq(bra_r(r),m,q) =
    prog(offset(atomofint(fetchr(r,q)),
    nextmemaddr(m)),q);
exq(if(o,r1,r2,m1),m,q) =
    prog(cond(applyrop(o,fetchr(r1,q),fetchr(r2,q)),
    m1,nextmemaddr(m)),q);
exq(ifi(o,r,v,m1),m,q) =
    prog(cond(applyrop(o,fetchr(r,q),v,
    m1,nextmemaddr(m)),q);
exq(ifte(o,r1,r2,m1,m2),m,q) =
    prog(cond(applyrop(o,fetchr(r1,q),fetchr(r2,q),
    m1,m2),q);
exq(iftei(o,r,v,m1,m2),m,q) =
    prog(cond(applyrop(o,fetchr(r,q),v,m1,m2),q);

exq(if_pcr(o,r1,r2,n),m,q) =
    prog(cond(applyrop(o,fetchr(r1,q),fetchr(r2,q),
    offset(n,nextmemaddr(m)),nextmemaddr(m)),q);
exq(ifi_pcr(o,r,v,n),m,q) =
    prog(cond(applyrop(o,fetchr(r,q),v),
    offset(n,nextmemaddr(m)),nextmemaddr(m)),q);
exq(ifte(o,r1,r2,i1,i2),m,q) =
    prog(cond(applyrop(o,fetchr(r1,q),fetchr(r2,q),
    offset(i1,nextmemaddr(m)),
    offset(i2,nextmemaddr(m))),q);
exq(iftei(o,r,v,m1,m2),m,q) =
    prog(cond(applyrop(o,fetchr(r,q),v),
    offset(i1,nextmemaddr(m)),
    offset(i2,nextmemaddr(m))),q);
exq(test(o,r1,m1),m,q) =
    prog(cond(applybop(o,fetchr(r1,q)),
    m1,nextmemaddr(m)),q);
exq(testm(o,m2,m1),m,q) =
    prog(cond(applybop(o,fetchm(m2,q)),
    m1,nextmemaddr(m)),q);
exq(teste(o,r1,m1,m2),m,q) =
```

```
        prog(cond(applybop(o,fetchr(r1,q)),
        m1,m2),q);
    exq(testme(o,m3,m1,m2),m,q) =
        prog(cond(applybop(o,fetchm(m3,q)),
        m1,m2),q);
    exq(test_pcr(o,r1,n),m,q) =
        prog(cond(applybop(o,fetchr(r1,q));
        offset(n,nextmemaddr(m)),
        nextmemaddr(m)),q);
    exq(testm_pcr(o,m2,n),m,q) =
        prog(cond(applybop(o,fetchm(m2,q));
        offset(n,nextmemaddr(m)),
        nextmemaddr(m)),q);
    exq(teste_pcr(o,r1,i1,i2),m,q) =
        prog(cond(applybop(o,fetchr(r1,q));
        offset(i1,nextmemaddr(m)),
        offset(i2,nextmemaddr(m))),q);
    exq(testme_pcr(o,m3,i1,i2),m,q) =
        prog(cond(applybop(o,fetchm(m3,q));
        offset(i1,nextmemaddr(m)),
        offset(i2,nextmemaddr(m))),q)
    exq(jsr(m1,s),m,q) =
        prog(m1,pushstk(
        valofmemaddr(nextmemaddr(m)),s,q));
    exq(jsr_i(m1,s),m,q) =
        prog(atomofmemaddr(fetchm(m1,q)),
        pushstk(valofmemaddr(nextmemaddr(m)),s,q));
    exq(jsr_i(r1,s),m,q) =
        prog(atomofmemaddr(fetchr(r1,q)),
        pushstk(valofmemaddr(nextmemaddr(m)),s,q));
    exq(bsr(n,s),m,q) =
        prog(offset(n,nextmemaddr(m)),
        pushstk(valofmemaddr(nextmemaddr(m)),s,q));
    exq(bsr_r(r,s),m,q) =
        prog(offset(atomofint(fetchr(r,q)),
        nextmemaddr(m)),
        pushstk(valofmemaddr(nextmemaddr(m)),s,q));
    exq(rts(s),m,q) =
        prog(atomofmemaddr(topstk(s,q)),popstk(s,q));
    exq(open(s),m,q) =
        prog(nextmemaddr(m),openfile(
            atomofstr.char(topstk(s,popstk(s,popstk(s,
                popstk(s,q))))),
            atomoffile(topstk(s,popstk(s,popstk(s,q)))),
            atomofint(topstk(s,popsstk(s,q))),
            atomofint(topstk(s,q),
            popstk(s,q)));
    exq(close(s),m,q) =
        prog(nextmemaddr(m),closefile(
            atomoffile(topstk(s,q)),
            popstk(s,q)));
    exq(read(s),m,q) =
```

```
            prog(nextmemaddr(m),storem(infile(
                atomoffile(topstk(s,popstk(s,q))),
                popstk(s,q)),
            atomofmemaddr(topstk(s,popstk(s,q)),
            popstk(s,q)));
        exq(write(s),m,q) =
            prog(nextmemaddr(m),outfile(fetchm(
                atomofmemaddr(topstk(s,popstk(s,q))),
                popstk(s,q));
            atomoffile(topstk(s,q)),
            popstk(s,q)));

end am;
```

APPENDIX B: COMPLETE SPECIFICATION OF A SUBSET OF THE ABSTRACT PROCESSOR

```
Resource intructions is

    Extension of
        natural,
        integer,
        memaddress,
        regaddress,
        stkaddress,
        operatorclasses,
        intructiontype,
        typing

    Operands

    Operators
        monad: mop,regaddr,regaddr -> instr;
        dyad: dop,regaddr,regaddr,regaddr -> instr;
        triad: top,regaddr,regaddr,regaddr,regaddr -> instr;
        mov_m_r: memaddr,regaddr -> instr;
        mov_r_m: regaddr,memaddr -> instr;
        mov_r_r: regaddr,regaddr -> instr;
        push_r: regaddr,stkaddr -> instr;
        pop_r: stkaddr,regaddr -> instr;
        jmp_r: regaddr -> instr;
        if: relop,regaddr,regaddr,memaddr -> instr;
        jsr: memaddr,stkaddr -> instr;
        rts: stkaddr -> instr;

    Properties

end intructions;


Resource amstate is

    Extension of
        boolean,
        natural,
        integer,
        str(character),
        memaddress,
        regaddress,
        files,
        identifiers,
        typing
```

```
Static
  Operands
      state;

  Operators
      fetchm: memaddr,state -> val;
      fetchr: regaddr,state -> val;
      storem: val,memaddr,state -> state;
      storer: val,regaddr,state -> state;
      topstk: stkaddr,state -> val;
      pushstk: val,stkaddr,state -> state;
      popstk: stkaddr,state -> state;
      initam: -> state;
      initstk: stkaddr,state -> state;

      active: memid,state -> bool;

  Properties
      topstk(s,initstk(s)) is undefined;
      popstk(s,initstk(s)) is undefined;
      popstk(s,initam()) is undefined;
      stateaxiom(m,memaddr);
      stateaxiom(r,regaddr);
       topstk(s(pushstk(v,s,q)) = v;
      popstk(s(pushstk(v,s,q)) = q;
      active(m,initam()) = false();
      active(m,storer(v,r,q)) = active(m,q);
      active(m,storem(v,a,q)) = active(m,q);
      active(m,initstk(s,q)) = active(m,q);
      active(m,pushstk(v,s,q) = active(m,q);
      active(m,popstk(s,q) = active(m,q);
      if active(m,q) = false()
      then
          fetchm(offset(n,m),q) is undefined;
      endif;
      if active(m,q) = false();
      then
          storem(v,offset(n,m),q) is undefined;
      endif;
      if ltint(n,ntoi(n2)) = true()
      then
          offset(n,offset(n1,startmemaddr(
              lalloc(n2,q)))) =
              offset(sumint(n,n1),
              startmemaddr(lalloc(n2,q)));
      else
          offset(n,offset(n1,startmemaddr(
              lalloc(n2,q)))) is undefined;
      endif;
      indir(zeronat(),m) = m;
      if whattype(fetchm(indir(n,m),q) = typememaddr()
      then
```

```
            indir(succnat(n),m) =
                atomofmemaddr(fetchm(indir(n,m),q));
        else
            indir(succnat(n),m) is undefined;
        endif;

Dynamic
  Entry Places
        req_fetchm(netlabel)[memaddr.state];
        req_storem(netlabel)[val.memaddr.state];

        req_fetchr(netlabel)[regaddr.state];
        req_storer(netlabel)[val.regaddr.state];

        req_topstk(netlabel)[stkaddr.state];
        req_pushstk(netabel)[val,stkaddr.state];
        req_popstk(netlabel)[stkaddr.state];
        req_initstk(netlabel)[stkaddr.state];

        req_initam()[]

  Exit Places
        avail_fetchm(netlabel)[val];
        avail_storem(netlabel)[state];

        avail_fetchr(netlabel)[val];
        avail_storer(netlabel)[state];

        avail_topstk(netlabel)[val];
        avail_pushstk(netlabel)[state];
        avail_popstk(netlabel)[state];
        avail_initstk(netlabel)[state];

        avail_initam[state];

  Internal Places
        access_avail[];
        fetchm_for(netlabel)[];
        fetchm_activated[memaddr.state];
        fetchm_completed[val];
        storem_for(netlabel)[];
        storem_activated[val.memaddr.state];
        storem_completed[state];

        access_avail[];
        fetchr_for(netlabel)[];
        fetchr_activated[regaddr.state];
        fetchr_completed[val];
        storer_for(netlabel)[];
        storer_activated[val.regaddr.state];
        storer_completed[state];
```

```
    accessk_avail[];
    topstk_for(netlabel)[];
    topstk_activated[stkaddr.state];
    topstk_completed[val];
    pushstk_for(netlabel)[];
    pushstk_activated[val.stkaddr.state];
    pushstk_completed[state];
    popstk_for(netlabel)[];
    popstk_activated[stkaddr.state];
    popstk_completed[state];
    initstk_for(netlabel)[];
    initstk_activated[stkaddr.state];
    initstk_completed[state];


Initial State
    => accessm_avail[];

    => accessr_avail[];

    => accessk_avail[];

Transitions
    act_fetchm: [memaddr.state],[] -> [memaddr.state],[];
    perform_fetchm: [memaddr.state] -> [val];
    finish_fetchm: [val],[] -> [val],[];
    act_storem: [val.memaddr.state],[] ->
        [val.memaddr.state],[];
    perform_storem: [val.memaddr.state] -> [state];
    finish_storem: [state],[] -> [state],[];

    act_fetchr: [regaddr.state],[] ->
        [regaddr.state],[];
    perform_fetchr: [regaddr.state] -> [val];
    finish_fetchr: [val],[] -> [val],[];
    act_storer: [val.regaddr.state],[] ->
        [val.regaddr.state],[];
    perform_storer: [val.regaddr.state] -> [state];
    finish_storer: [state],[] -> [state],[];

    act_topstk: [stkaddr.state],[] ->
        [stkaddr.state],[];
    perform_topstk: [stkaddr.state] -> [val];
    finish_topstk: [val],[] -> [val],[];
    act_pushstk: [val.stkaddr.state],[] ->
        [val.stkaddr.state],[];
    perform_pushstk: [val.stkaddr.state] -> [state];
    finish_pushstk: [state],[] -> [state],[];
    act_popstk: [stkaddr.state],[] ->
        [stkaddr.state],[];
    perform_popstk: [stkaddr.state] -> [state];
    finish_popstk: [state],[] -> [state],[];
```

```
    act_initstk: [stkaddr.state],[] ->
        [stkaddr.state],[];
    perform_initstk: [stkaddr.state] -> [state];
    finish_initstk: [state],[] -> [state],[];

    perform_initam: [] -> [state];

Properties
    act_fetchm(req_fetchm(l)[m.q], accessm_avail[]) =>
        fetchm_for(l)[],fetchm_activated[m.q];
    perform_fetchm(fetchm_activated[m.q]) =>
        fetchm_completed[v];
    finish_fetchm(fetch_completed[v], fetchm_for(l)[])
         => avail_fetchm(l)[v], accessm_avail[];
    act_storem(req_storem(l)[v.m.q], accessm_avail[]) =>
        storem_for(l)[], storem_activated[v.m.q];
    perform_storem(storem_activated[v.m.q]) =>
        storem_completed[q];
    finish_storem(storem_completed[q],
        storem_for(l)[]) =>
        avail_storem(l)[q], accessm_avail[];

    act_fetchr(req_fetchr(l)[.q], accessr_avail[]) =>
        fetchr_for(l)[],fetchr_activated[.q];
    perform_fetchr(fetchr_activated[.q]) =>
        fetchr_completed[v];
    finish_fetchr(fetch_completed[v], fetchr_for(l)[])
        => avail_fetchr(l)[v], accessr_avail[];
    act_storer(req_storer(l)[v..q], accessr_avail[])
        => storer_for(l)[], storer_activated[v..q];
    perform_storer(storer_activated[v..q]) =>
        storer_completed[q];
    finish_storer(storer_completed[q],
        storer_for(l)[]) => avail_storer(l)[q],
        accessr_avail[];

    act_topstk(req_topstk(l)[s.q], accessk_avail[]) =>
        topstk_for(l)[], topstk_activated[s.q];
    perform_topstk(topstk_activated[s.q]) =>
        topstk_completed[v];
    finish_topstk(topstk_completed[v],topstk_for(l)[] =>
        avail_topstk(l)[v];
    act_pushstk(req_pushstk(l)[v.s.q], accessk_avail[])
        => pushstk_for(l)[], pushstk_activated[v.s.q];
    perform_pushstk(pushstk_activated[s.q]) =>
        pushstk_completed[q1];
    finish_pushstk(pushstk_completed[q1],pushstk_for(l)[]
        => avail_pushstk(l)[q1];
    act_popstk(req_popstk(l)[s.q], accessk_avail[]) =>
        popstk_for(l)[], popstk_activated[s.q];
    perform_popstk(popstk_activated[s.q]) =>
        popstk_completed[q1];
```

```
        finish_popstk(popstk_completed[q1],popstk_for(l)[] =>
            avail_popstk(l)[q1];
        act_initstk(req_initstk(l)[s.q], accessk_avail[]) =>
            initstk_for(l)[], initstk_activated[s.q];
        perform_initstk(initstk_activated[s.q]) =>
            initstk_completed[q1];
        finish_initstk(initstk_completed[q1],initstk_for(l)[]
            => avail_initstk(l)[q1];

        perform_initam(req_initam[]) => avail_initam[state];

end amstate;


Resource am is

    Extension of
        memaddress,
        intructiontype,
        typing,
        amstate

    Operands
        prog: memaddr,state -> state;

        cond: val,memaddr,memaddr - memaddr;
        exq: instr,memaddr,state -> state;

    Operators
        prog(m,q) = exq(atomofinstr(fetchm(m,q)),m,q);
        cond(valofbool(true(),m,m2)) = m;
        cond(valofboll(false(),m,m2)) = m2;

    Properties
        exq(monad(o,r,r2),m,q) =
            prog(nextmemaddr(m),
                storer(applymop(o,fetchr(r,q)),
                    r2,q));
        exq(dyad(o,r,r2,r3),m,q) =
            prog(nextmemaddr(m),
                storer(applydop(o,
                    fetchr(r,q),fetchr(r2,q)),r3,q));
        exq(triad(o,r,r2,r3,r4),m,q) =
            prog(nextmemaddr(m),
                storer(applytop(o,fetchr(r,q),
                    fetchr(r2,q),fetchr(r3,q)),r4,q));
        exq(mov_m_r(m,r),m,q) =
            prog(nextmemaddr(m),
                storem(fetchm(m,q),r,q));
        exq(mov_r_m(r,m),m,q) =
            prog(nextmemaddr(m),
                storem(fetchr(r,q),m,q));
```

```
        exq(mov_r_r(r,r2),m,q) =
            prog(nextmemaddr(m),
                storem(fetchr(r,q),r2,q));
        exq(push_r(r,s),m,q) =
            prog(nextmemaddr(m),
                pushstk(fetchr(r,q),s,q));
        exq(pop_r(s,r),m,q) =
            prog(nextmemaddr(m),
                popstk(s,storer(topstk(s,q),r,q))),m,q);
        exq(jmp_r(r),m,q) =
            prog(atomofmemaddr(fetchr(r,q)),q);
        exq(if(o,r,r2,m1),m,q) =
            prog(cond(applyrop(o,fetchr(r,q),fetchr(r2,q)),
            m1,nextmemaddr(m)),q);
        exq(jsr(m,s),m,q) =
            prog(m,pushstk(
            valofmemaddr(nextmemaddr(m)),s,q));
        exq(rts(s),m,q) =
            prog(atomofmemaddr(topstk(s,q)),popstk(s,q));

Dynamic
  Entry Places
        req_prog[memaddr.state];

        req_exq(netlabel)[instr.memaddr.state];

        req_cond(netlabel)[val.memaddr.memaddr];

  Exit Places
        avail_exq(netlabel)[memaddr.state];

        avail_cond(netlabel)[memaddr];

  Internal Places
        prog_avail[];
        prog_fetch[memaddr.state];
        prog_instr[memaddr.state];
        prog_perform[]

        exq_avail[];
        exq_for(netlabel)[];

        exq_monad_activated[state];
        exq_monad_fetch[state];
        exq_monad_apply[state];
        exq_monad_store[];

        exq_dyad_activated[state];
        exq_dyad_fetch[state];
        exq_dyad_apply[state];
        exq_dyad_store[];
```

```
    exq_triad_activated[state];
    exq_triad_fetch[state];
    exq_triad_apply[state];
    exq_triad_store[];

    exq_mov_r_r_activated[state];
    exq_mov_r_r_perform[state];
    exq_mov_r_r_store[];

    exq_mov_r_m_activated[state];
    exq_mov_r_m_perform[state];
    exq_mov_r_m_store[];

    exq_mov_m_r_activated[state];
    exq_mov_m_r_perform[state];
    exq_mov_m_r_store[];

    exq_push_r_activated[state];
    exq_push_r_perform[state];
    exq_push_r_push[];

    exq_pop_r_activated[state];
    exq_pop_r_perform[state];
    exq_pop_r_store[];
    exq_pop_r_pop[];

    exq_jmp_r_activated[state];
    exq_jmp_r_perform[state];
    exq_jmp_r_converting[state];

    exq_if_activated[state];
    exq_if_fetch[state];
    exq_if_cond[state];

    cond_activated[memaddr.memaddr];

Initial State
    => prog_avail[];

    => exq_avail[];

    => cond_avail[];

Transitions
activate_prog: [],[memaddr.state] ->
    [memaddr.state],[memaddr.state];
get_instr_prog: [memaddr.state],[val] ->
    [memaddr.state],[val];
perform_prog: [memaddr.state],[instr] ->
    [],[instr.memaddr.state];
finish_prog: [],[memaddr.state] ->
    [],[memaddr.state];
```

```
activate_exq_monad: [instr.memaddr.state],[] ->
    [state],[instr],[instr],[instr],[memaddr];
start_exq_monad: [state],[regaddr], ->
    [state],[regaddr.state];
apply_exq_monad: [state],[operator],[val] ->
    [state],[operator.val];
store_exq_monad: [state],[val],[operator] ->
    [],[val.regaddr.state];
finish_exq_monad: [],[state],[memaddr] ->
    [memaddr.state];

activate_exq_dyad: [instr.memaddr.state],[] ->
    [state],[instr],[instr],[instr],[instr],[memaddr];
start_exq_dyad: [state],[regaddr],[regaddr] ->
    [state],[regaddr.state],[regaddr.state];
apply_exq_dyad: [state],[operator],[val],[val] ->
    [state],[operator.val,val];
store_exq_dyad: [state],[val],[regaddr] ->
    [],[val.regaddr.state];
finish_exq_dyad: [],[state],[memaddr] ->
    [memaddr.state];

activate_exq_triad: [instr.memaddr.state],[] ->
    [state],[instr],[instr],[instr],[instr],
    [instr],[memaddr];
start_exq_triad: [state],[regaddr],[regaddr],[regaddr] ->
    [state],[regaddr.state],[regaddr],[regaddr];
apply_exq_triad: [state],[operator],[val],[val],[val] ->
    [state],[operator.val.val.val];
store_exq_triad: [state],[val],[regaddr] ->
    [],[val.regaddr.state];
finish_exq_triad: [],[state],[memaddr] ->
    [memaddr.state];

activate_exq_mov_r_r: [instr.memaddr.state],[] ->
    [state],[instr],[instr],[memaddr];
start_exq_mov_r_r: [start],[regaddr] ->
    [state],[regaddr.state];
store_exq_mov_r_r: [state],[regaddr],[val] ->
    [],[val.regaddr.state];
finish_exq_mov_r_r: [],[state],[memaddr] ->
    [memaddr.state];

activate_exq_mov_r_m: [instr.memaddr.state],[] ->
    [state],[instr],[instr],[memaddr];
start_exq_mov_r_m: [start],[regaddr] ->
    [state],[regaddr.state];
store_exq_mov_r_m: [state],[memaddr],[val] ->
    [],[val.memaddr.state];
finish_exq_mov_r_m: [],[state],[memaddr] ->
    [memaddr.state];
```

```
activate_exq_mov_m_r: [instr.memaddr.state],[] ->
    [state],[instr],[instr],[memaddr];
start_exq_mov_m_r: [start],[memaddr] ->
    [state],[regaddr.state];
store_exq_mov_m_r: [state],[regaddr],[val] ->
    [],[val.regaddr.state];
finish_exq_mov_m_r: [],[state],[memaddr] ->
    [memaddr.state];

activate_exq_push_r: [instr.memaddr.state],[] ->
    [state],[instr],[inestr],[memaddr];
fetch_exq_push_r: [state],[regaddr] ->
    [state],[regaddr.state];
push_exq_push_r: [state],[val],[stkaddr] ->
    [],[stkaddr.state];
finish_exq_push_r: [],[state],[memaddr] ->
    [memaddr.state];

activate_exq_pop_r: [instr.memaddr.state],[] ->
    [state],[instr],[instr],[memaddr];
pop_exq_pop_r: [state],[stkaddr] ->
    [state],[stkaddr.state];
store_exq_pop_r: [state],[val],[regaddr] ->
    [stkaddr],[stkaddr.state];
top_exq_pop_r: [stkaddr],[state] ->
    [],[stkaddr.state];
finish_exq_pop_r: [],[state],[memaddr] ->
    [memaddr.state];

activate_exq_jmp_r([instr.memaddr.state],[]) ->
    [state],[instr];
fetch_exq_jmp_r([state],[regaddr] ->
    [state],[regaddr.state];
convert_exq_jmp_r([state],[val]) ->
    [state],[val];
finish([state],[memaddr]) ->
    [memaddr.state];

activate_cond([],[val.memaddr.memaddr]) ->
    [memaddr.memaddr],[val];
finish_cond([memaddr.memaddr],[bool]) ->
    [memaddr];

activate_exq_if([],[instr.memaddr.state]) ->
    [],[state],[instr],[instr],[instr],[memaddr];
start_exq_if([state],[regaddr],[regaddr]) ->
    [regaddr.state],[regaddr.state];
apply_exq_if([state],[val],[val],[operator]) ->
    [state],[operator.val.val];
cond_exq_if([state],[memaddr],[val],[memaddr]) ->
    [state],[val.memaddr.memaddr];
```

```
finish_exq_if([state],[memaddr] ->
    [memaddr.state];

Properties
activate_prog(prog_avail[], req_prog[m.q]) =>
    prog_fetch[m.q], req_fetchm(11)[m.q];
get_instr_prog(prog_activated[m.q], avail_fetchm(11)[v])
    => prog_instr[m.q], req_atomofinstr(12)[v];
perform_prog(prog_instr[m.q],
    avail_atomofinstr(12)[i]) =>
    prog_perform[], req_exq(13)[i.m.q];
finish_prog(prog_perform[], avail_exq[m1.q1]) =>
    prog_avail[], req_prog[m1.q1];

activate_exq_monad(exq_avail[],
    req_exq(1)[monad(o.r1.r2).m.q]) =>
    exq_for(1)[], exq_monad_activated[q],
    req_operator(11)[monad(o.r1.r2)],
    req_operand1(12)[monad(o.r1.r2)],
    req_operand2(13)[monad(o.r1.r2)],
    req_nextmemaddr(14)[m];
start_exq_monad(exq_monad_activated[q],
    avail_operand1(11)[r1]) -> exq_monad_fetch[q],
    req_fetchr(15)[r1.q];
apply_exq_monad(exq_monad_fetch[q], avail_fetchr(15)[v],
    avail_operator(11)[o]) => exq_monad_apply[q],
    req_apply_mop(16)[o.v];
store_exq_monad(exq_monad_apply[q],
    avail_apply_mop(16)[v1], avail_operand2(13)[r2]) =>
    exq_monad_store[], req_storer(17)[v1.r2.q];
finish_exq_monad(exq_monad_store[], avail_storer(17)[q1],
    avail_nextmemaddr(14)[m1]) => avail_exq(1)[m1.q1];

activate_exq_dyad(exq_avail[],
    req_exq(1)[dyad(o.r1.r2,r3).m.q]) =>
    exq_for(1)[], exq_dyad_activated[q],
    req_operator(11)[dyad(o.r1.r2,r3)],
    req_operand1(12)[dyad(o.r1.r2,r3)],
    req_operand2(13)[dyad(o.r1.r2,r3)],
    req_operand3(18)[dyad(o.r1.r2,r3)],
    req_nextmemaddr(14)[m];
start_exq_dyad(exq_dyad_activated[q],
    avail_operand1(11)[r1]),avail_operand2(12)[r2]
    -> exq_dyad_fetch[q],req_fetchr(15)[r1.q]
    req_fetch(19)[r2.q];
apply_exq_dyad(exq_dyad_fetch[q], avail_fetchr(15)[v1],
    avail_operator(11)[o]),avail_fetchr(19)[v2]
     => exq_dyad_apply[q],req_apply_dop(16)[o.v1.v2];
store_exq_dyad(exq_dyad_apply[q],
    avail_apply_dop(16)[v3], avail_operand3(18)[r3]) =>
    exq_dyad_store[], req_storer(17)[v3.r3.q];
```

```
    finish_exq_dyad(exq_dyad_store[], avail_storer(17)[q1],
        avail_nextmemaddr(14)[m1]) => avail_exq(1)[m1.q1];

    activate_exq_triad(exq_avail[],
        req_exq(1)[triad(o.r1.r2,r3,r4).m.q]) =>
        exq_for(1)[], exq_triad_activated[q],
        req_operator(11)[triad(o.r1.r2,r3)],
        req_operand1(12)[triad(o.r1.r2,r3)],
        req_operand2(13)[triad(o.r1.r2,r3)],
        req_operand3(18)[triad(o.r1.r2,r3)],
        req_operand4(110)[triad(o.r1.r2,r3,r4)],
        req_nextmemaddr(14)[m];
    start_exq_triad(exq_triad_activated[q],
        avail_operand1(11)[r1]),avail_operand2(12)[r2]
        avail_operand3(13)[r3],-> exq_triad_fetch[q],
        req_fetchr(15)[r1.q],req_fetch(19)[r2.q],
        req_fetchr(111)[r3.q];
    apply_exq_triad(exq_triad_fetch[q], avail_fetchr(15)[v1],
        avail_operator(11)[o]),avail_fetchr(19)[v2],
        avail_fetchr(111)[v3] => exq_triad_apply[q],
        req_apply_top(16)[o.v1.v2.v3];
    store_exq_triad(exq_triad_apply[q],
avail_apply_top(16)[v4], avail_operand4(18)[r4]) =>
        exq_triad_store[], req_storer(17)[v4.r4.q];
    finish_exq_triad(exq_triad_store[], avail_storer(17)[q1],
        avail_nextmemaddr(14)[m1]) => avail_exq(1)[m1.q1];

    activate_exq_mov_r_r(exq_avail[],
        req_exq(1)[mov_r_r(r1,r2).m.q]) =>
        exq_mov_r_r_activated[q],
        req_operand1(11)[mov_r_r(r1,r2)],
        req_operand2(12)[mov_r_r(r1,r2)],
        req_nextmemaddr(13)[m];
    start_exq_mov_r_r(exq_mov_r_r_activated[q],
        avail_operand1(11)[r1]) =>
        exq_mov_r_r_perform[q], req_fetchr(14)[r1.q];
    store_exq_mov_r_r(exq_mov_r_r_perform[q],
        avail_fetchr(14)[v], avail_operand2(12)[r2]) =>
        exq_mov_r_r_store[], req_storer[v.r2.q];
    finish_exq_mov_r_r(exq_mov_r_r_store[], avail_storer[q1],
        avail_nextmemaddr(13)[m1]) =>
        avail_exq(1)[m1.q1];

    activate_exq_mov_r_m(exq_avail[],
        req_exq(1)[mov_r_m(r1,m2).m.q]) =>
        exq_mov_r_m_activated[q],
        req_operand1(11)[mov_r_m(r1,m2)],
        req_operand2(12)[mov_r_m(r1,m2)],
        req_nextmemaddr(13)[m];
    start_exq_mov_r_m(exq_mov_r_m_activated[q],
        avail_operand1(11)[r1]) =>
        exq_mov_r_m_perform[q], req_fetchr(14)[r1.q];
```

```
store_exq_mov_r_m(exq_mov_r_m_perform[q],
    avail_fetchr(l4)[v], avail_operand2(l2)[m2]) =>
    exq_mov_r_m_store[], req_storem[v.m2.q];
finish_exq_mov_r_m(exq_mov_r_m_store[], avail_storem[q1],
    avail_nextmemaddr(l3)[m1]) =>
    avail_exq(l)[m1.q1];

activate_exq_mov_m_r(exq_avail[],
    req_exq(l)[mov_m_r(m2,r2).m.q]) =>
    exq_mov_m_r_activated[q],
    req_operand1(l1)[mov_m_r(m2,r2)],
    req_operand2(l2)[mov_m_r(m2,r2)],
    req_nextmemaddr(l3)[m];
start_exq_mov_m_r(exq_mov_m_r_activated[q],
    avail_operand1(l1)[m2]) =>
    exq_mov_m_r_perform[q], req_fetchm(l4)[m2.q];
store_exq_mov_m_r(exq_mov_m_r_perform[q],
    avail_fetchm(l4)[v], avail_operand2(l2)[r2]) =>
    exq_mov_m_r_store[], req_storer[v.r2.q];
finish_exq_mov_m_r(exq_mov_m_r_store[], avail_storer[q1],
    avail_nextmemaddr(l3)[m1]) =>
    avail_exq(l)[m1.q1];

activate_exq_push_r(exq_avail[],
    req_exq(l)[push_r(s,r).m.q]) =>
    exq_push_r_activated[q],
    req_operand1(l1)[push_r(s,r)],
    req_operand2(l2)[push_r(s,r)],
    req_nextmemaddr(l3)[m];
fetch_exq_push_r(exq_push_r_activated[q],
    avail_operand2(l2)[r]) =>
    exq_push_r_perform[q], req_fetchr(l4)[r.q];
push_exq_push_r(exq_push_r_perform[q],
    avail_fetchr(l4)[v], avail_operand1(l1)[s]) =>
    exq_push_r_store[], req_push[v.s.q];
finish_exq_push_r(exq_push_r_store[], avail_push[q1],
    avail_nextmemaddr(l3)[m1]) =>
    avail_exq(l)[m1.q1];

activate_exq_pop_r(exq_avail[],
    req_exq(l)[pop_r(s,r).m.q]) =>
    exq_pop_r_activated[q],
    req_operand1(l1)[pop_r(s,r)],
    req_operand2(l2)[pop_r(s,r)],
    req_nextmemaddr(l3)[m];
pop_exq_pop_r(exq_pop_r_activated[q],
    avail_operand1(l1)[s]) =>
    exq_pop_r_perform[q], req_top(l4)[s.q];
store_exq_pop_r(exq_pop_r_perform[q],
    avail_top(l4)[v], avail_operand2(l2)[r]) =>
    exq_pop_r_store[s], req_storer[v.r.q];
top_exq_pop_r(exq_pop_r_store[s],avail_storer[q1] =>
```

```
            exq_pop_r_pop[],req_pop[s.q1];
    finish_exq_pop_r(exq_pop_r_pop[], avail_pop[q2],
        avail_nextmemaddr(13)[m1]) =>
        avail_exq(1)[m1.q2];

    activate_exq_jmp_r(exq_avail[],
        req_exq(1)[jmp(r).m.q]) =>
        exq_jmp_r_activated[q],
        req_operand1(11)[jmp(r)];
    fetch_exq_jmp_r(exq_jmp_r_activated[q],
        avail_operand1(11)[r] =>
        exq_jmp_r_perform[q],req_fetchr(12)[r.q];
    convert_exq_jmp_r(exq_jmp_r_perform[q],
        avail_fetchr(12)[v]) =>
        exq_jmp_r_converting[q],req_atomofmemaddr(13)[v];
    finish_exq_jmp_r(exq_jmp_r_converting[q],
        avail_atomofmemaddr(13)[m1] =>
        avail_exq(1)[m1.q];

    activate_cond(cond_avail[],req_cond(1)[v.m1.m2]) =>
        cond_activated[m1.m2],req_atomofbool(11)[v];
    finish_cond(cond_activated[m1.m2],
        avail_atomofbool(11)[true()] =>
        avail_cond(1)[m1];
    finish_cond(cond_activated[m1.m2],
        avail_atomofbool(11)[false()] =>
        avail_cond(1)[m2];

    activate_exq_if(exq_avail[],
        req_exq(1)[if(o.r1.r2.m1).m.q]) =>
        exq_for(1)[], exq_if_activated[q],
        req_operator(11)[if(o.r1.r2.m1)],
        req_operand1(12)[if(o.r1.r2.m1)],
        req_operand2(13)[if(o.r1.r2.m1)],
        req_nextmemaddr(14)[m]
    start_exq_if(exq_if_activated[q],
avail_operand1(11)[r1],avail_operand2(12)[r2])
        -> exq_if_fetch[q],req_fetchr(15)[r1.q]
        req_fetchr(17)[r1.q];
    apply_exq_if(exq_if_fetch[q], avail_fetchr(15)[v1],
        avail_fetchr(17)[v2], avail_operator(11)[o]) =>
        exq_if_apply[q], req_apply_rop(16)[o.v1.v2];
    cond_exq_if(exq_if_apply[q],avail_nextmemaddr(14)[m2]
        avail_operand3(13)[m3],avail_apply_rop(16)[v3], =>
        exq_if_cond[q], req_cond(17)[v3.m3.m2];
    finish_exq_if(exq_if_cond[q], avail_cond(17)[m3]) =>
        avail_exq(1)[m3.q];

end am;
```

LIST OF REFERENCES

Bergstra, A. and Tucker, J, V., "Initial and Final Algebra Semantics for Data Type Specifications: Two Characterizaaion Theorems", SIAM Journal of Computing, Vol. 12, No. 2, May 1983.

Davis, D., Tech. Report NPS52 84-022, A Formal Method for Specifying Computer Resources in an Implementation Independent Manner, Naval Postgraduate School, Monterey, California, December 1984.

Davis, D. and Yurchak, J. M., The Specification, Design, and Implementaion of an Abstrct Processor, paper presented at the Alisomar Conference on Circuits, Systems and Computers, 19th, Pacific Grove, California, 6-8 November 1985.

Goguen, J. A.,Thatcher, J. W., and Wagner, E. G., "An Initial Algebra to the Specification, Correctness, and Implementation of Abstrcat Data Types", in Current Trends in Programming Methodology IV, Data Structuring, ed.,R. T. Yeh, Prentice-Hall, Englewood Cliffs, N.J., 1978, pp. 80-149.

Guttag, J. V., Horowitz, E. and Musser, D. R., "Abstract Type Specifications", in Current Trends in Programming Methodology IV, Data Structuring, ed., R. T. Yeh, Prentice-Hall, Englewood Cliffs, N.J., 1978, pp. 60-79.

Hunter, J.E., The Formal Specification of a Visual Display device: Design and Implementation, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1985.

Jensen, K. and Schmidt, E. M., "Pascal Semantics by a Combination of Denotional Semantics and High_Level Petri Nets", in Lecture Notes in Computer Science, No. 222: Advances in Petri Nets 1985, eds., G. Goos and J. Hartmanis, 1986, pp. 297-329.

Krämer, B., "A Formal and Semigraphical Language Combining Petri Nets and Abstrct Data Types for the Specification of Distributed Systems", ACM, 0270-5257/87/0300/0116, pp. 116-125, 1987.

Peterson, J. L., Petri Net Theory and the Modeling of Systems, Prentice-Hall, Englewood Cliffs, N.J., 1981.

Petri, C., Kommunikation mit Automaten, Ph.D. Dissertation, University of Bonn, Bonn, West Germany, 1962.

Yurchak, J. M., The Formal Specification of an Abstract Machine: Design and Implementation, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1984.

Zang, K. H., The Formal Specification of an Abstract Database: Design and Implementation, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1985.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
4.	Curriculum Officer, Code 37 Computer Technology Programs Naval Postgraduate School Monterey, California 93943-5000	1
5.	Daniel Davis, Code 52Vv Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	5
6.	Valdis Berzins, Code 52Be Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
7.	Kommando Marineführungssysteme Heppenser Groden 2940 Wilhelmshaven West-Germany	1
8.	OltzS Klaus Karrasch Kommando Marineführungssysteme Heppenser Groden 2940 Wilhelmshaven West-Germany	5
9.	Robert Westbrook, Code 31C Naval Weapons Center China Lake, California 93555	1
10.	Robert Wasilausky Naval Ocean Systems Center 271 Catalina Boulevard San Diego, California 92152	1

Thesis
K14485
c.1

Karrasch
The formal specification of computer systems using Petri Nets.